

# Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/JP05/000451

International filing date: 11 January 2005 (11.01.2005)

Document type: Certified copy of priority document

Document details: Country/Office: JP  
Number: 2004-008327  
Filing date: 15 January 2004 (15.01.2004)

Date of receipt at the International Bureau: 24 February 2005 (24.02.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland  
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

11.01.2005

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日                      2 0 0 4 年   1 月 1 5 日  
Date of Application:

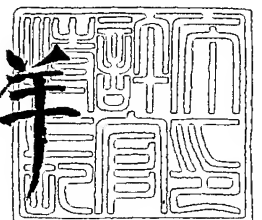
出 願 番 号                      特 願 2 0 0 4 - 0 0 8 3 2 7  
Application Number:  
[ST. 10/C]:                      [ J P 2 0 0 4 - 0 0 8 3 2 7 ]

出 願 人                      松 下 電 器 産 業 株 式 会 社  
Applicant(s):

2 0 0 5 年   2 月 1 4 日

特許庁長官  
Commissioner,  
Japan Patent Office

小 川 洋



【書類名】 特許願  
【整理番号】 2038250005  
【提出日】 平成16年 1月15日  
【あて先】 特許庁長官殿  
【国際特許分類】 G06F 1/00 340  
【発明者】  
    【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内  
    【氏名】 舘上 竜司  
【特許出願人】  
    【識別番号】 000005821  
    【氏名又は名称】 松下電器産業株式会社  
【代理人】  
    【識別番号】 100097179  
    【弁理士】  
    【氏名又は名称】 平野 一幸  
【手数料の表示】  
    【予納台帳番号】 058698  
    【納付金額】 21,000円  
【提出物件の目録】  
    【物件名】 特許請求の範囲 1  
    【物件名】 明細書 1  
    【物件名】 図面 1  
    【物件名】 要約書 1  
    【包括委任状番号】 0013529

**【書類名】 特許請求の範囲****【請求項 1】**

メモリのアクセス違反を検出しながら通常の処理を行うステップと、アクセス違反が検出されたら、アドレスに関連付けられた範囲情報と障害復帰情報とに基づいて復帰が可能か否かを判定するステップと、

復帰が可能なら障害復帰を行うステップとを含む情報処理方法。

**【請求項 2】**

復帰が可能でないなら停止処理を行うステップをさらに含む請求項 1 記載の情報処理方法。

**【請求項 3】**

障害復帰情報は、固定サイズ属性と可変サイズ属性とを有する請求項 1 から 2 記載の情報処理方法。

**【請求項 4】**

可変サイズ属性は、上方伸長属性と下方伸長属性とを有する請求項 3 記載の情報処理方法。

**【請求項 5】**

前記停止処理を行うステップにおいて、保存可能なデータがあれば、このデータを保存する請求項 2 記載の情報処理方法。

**【請求項 6】**

前記障害復帰を行うステップにおいて、読み／書きのアクセス種別を判定し、判定結果に基づいて異なる内容の障害復帰を行う請求項 1 から 5 記載の情報処理方法。

**【請求項 7】**

アクセス種別が「書き」であるとき、そのまま障害復帰し、アクセス種別が「読み」であるとき読み出し先に所定値を格納して障害復帰する請求項 6 記載の情報処理方法。

**【請求項 8】**

前記判定するステップにおいて、上方伸長属性があり、かつ、アクセス違反が下方である場合、復帰は可能でないと判定する請求項 4 記載の情報処理方法。

**【請求項 9】**

前記判定するステップにおいて、下方伸長属性があり、かつ、アクセス違反が上方である場合、復帰は可能でないと判定する請求項 4 又は 8 記載の情報処理方法。

**【請求項 10】**

前記判定するステップにおいて、固定サイズ属性がある場合、復帰は可能でないと判定する請求項 4、8 又は 9 記載の情報処理方法。

**【請求項 11】**

前記障害復帰を行うステップにおいて、アクセス違反した領域とは別の領域を確保し、確保した別の領域にアクセスする請求項 1 から 10 記載の情報処理方法。

**【請求項 12】**

障害復帰情報は、ターミネータ属性を有し、ターミネータ属性がデータの終端にターミネート値を有することを示す場合、前記障害復帰を行うステップにおいて、データの末尾にターミネート値を付加して障害復帰する請求項 1 から 11 記載の情報処理方法。

**【請求項 13】**

メモリと、

アドレス、範囲情報及び障害復帰情報が格納されるアドレスレジスタと、

前記アドレスレジスタを参照して前記メモリのアクセス違反を検出するアクセス違反検出部と、

前記メモリをアクセスし通常の処理を行う通常処理部と、

前記違反検出部がアクセス違反を検出したら前記アドレスレジスタを参照して復帰が可能か否かを判定する障害復帰判定部と、

前記障害復帰判定部が復帰が可能と判定すると、障害復帰を行う復帰処理部とを備える情報処理装置。

**【請求項 14】**

前記障害復帰判定部が復帰が可能でないと判定すると、停止処理を行う停止処理部をさらに備える請求項 13 記載の情報処理装置。

**【請求項 15】**

請求項 13 又は 14 記載の情報処理装置と、前記情報処理装置と障害復帰情報を通信する情報センターとを備える障害情報監視システム。

**【請求項 16】**

アドレス、範囲情報及び障害復帰情報が格納されるアドレスレジスタと、

前記アドレスレジスタを参照して外部のメモリをアクセスする命令処理部と、前記アドレスレジスタを参照し前記命令処理部が前記メモリに対しアクセス違反したことを検出すると前記命令処理部に例外信号を出力するアクセス違反検出部とを備えるプロセッサ。

**【請求項 17】**

ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする方法であって、

変数領域を指すポインタ値に、変数領域のアドレスと変数領域の範囲情報とアクセス違反が発生した際の障害復帰情報とを設定するステップを含むコンパイル方法。

**【請求項 18】**

ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする方法であって、

変数領域を指すポインタ値に、変数領域のアドレスと変数領域の範囲情報とを設定するステップと、

このポインタ値により指される変数領域に、アクセス違反が発生した際の障害復帰情報を設定するステップとを含むコンパイル方法。

**【請求項 19】**

障害復帰情報は、ソースコードとは別個に管理される請求項 17 から 18 記載のコンパイル情報。

**【請求項 20】**

ソースコードに障害復帰情報を付加した中間コードを生成するステップを含む請求項 17 から 19 記載のコンパイル情報。

**【請求項 21】**

ポインタを扱えるプログラム言語で記述されたソースコードをコンパイルする言語解析部と、

前記言語解析部が出力するコードに、変数に対応する障害復帰情報のコードを追加するコード生成部とを備えるコンパイル装置。

## 【書類名】明細書

【発明の名称】情報処理方法、その装置、それを用いた障害情報監視システム、プロセッサ、コンパイル方法及びその装置

## 【技術分野】

## 【0001】

本発明は、障害復帰機能を強化した情報処理装置及びその関連技術に関するものである。

## 【背景技術】

## 【0002】

近年、組み込み機器向けのプログラムを、C言語を用いて開発する機会が増えており、このようなプログラムの規模は、著しく増大する傾向にある。従って、バグのまったく存在しないプログラムを短期間で作成することは困難になっており、運用の段階でバグが顕在化して損害を発生する可能性が増大している。

## 【0003】

運用段階での顕在化の多いバグとして、バッファオーバーランによるセキュリティーホールバグが挙げられる。

## 【0004】

攻撃者は、プログラム実行時にスタック領域に確保された自動変数領域に、不正なデータを読み込ませ、システムダウンや、システムクラッシュを起こすことがある。

## 【0005】

以下、図22を用いて、バッファオーバーラン攻撃を説明する。

## 【0006】

図22(a)に示すソースコードでは、関数 `read__int` が内部において関数 `read__str` を呼出す。関数 `read__str` は、外部受信データを解析して、配列 `buf` に文字列を返す。関数 `read__int` は、関数 `atoi` を呼出し、得られた文字列を数値に変換し、変換後の数値を戻り値とする。

## 【0007】

より詳しくは、関数 `read__int` は、図22(b)に示すようにスタック上に確保された配列 `buf` 内に文字列を受け取り、ANSI標準関数 `atoi` が配列 `buf` 内の文字列を数値に変換し、その数値がリターンパラメータ用のレジスタに格納され、スタック上に存在する呼び出し元のアドレスの参照により、利短処理が実行される。

## 【0008】

ここで、関数 `read__str` の仕様として、32バイト以上の文字列を返さないものとする。

## 【0009】

ところが、関数 `read__str` 内部のロジックが複雑であり、特殊なデータが送付されると、32バイトを超える任意のデータが、関数 `read__int` の配列 `buf` に返されてしまうバグが潜在していたと仮定する。

## 【0010】

このような場合、攻撃者は、図22(c)に示すように、攻撃コードのアドレスが関数 `read__int` のスタックに返されるようなデータを作成して送りつけることができる。

## 【0011】

この場合、関数 `read__int` が関数 `read__str` を呼出した後、スタックに攻撃コードのアドレスが上書きされる。そして、関数 `atoi` がダミー部分を数値に変換した後、関数 `read__int` は呼び出し元関数ではなく攻撃コードにリターンしてしまう。

## 【0012】

こうなると、攻撃者は、攻撃コードにより任意のコードを実行可能となる。また、任意のコード実行ができなくとも、リターンアドレスが不正になるとプログラムは暴走し、シ

システムダウンが誘発されるおそれ大きい。

【0013】

現時点において主たる攻撃対象となっているのは、ネットワークに接続されたサーバー装置やパーソナルコンピュータ装置であり、システムのクラッキング、ウイルスの感染経路、DoS (denial of service attack) の踏み台などの問題が頻発している。

【0014】

また近い将来、ネットワークに接続される組み込み機器（例えば、携帯電話やネットワーク対応家電品など）は、ますます増加するものと予想される。

【0015】

組み込み機器では、ハードウェア制御や高速化のためC言語利用率が極めて高く、且つ、C言語を利用すると障害時においてプログラムを修正することは困難である。

【0016】

また、ネットワークに接続するために同一のソフトウェアモジュールが多用されるため、一度セキュリティホールが発覚した場合、攻撃者は、例えばルートDNSサーバーに正規リクエストを繰り返す自己増殖プログラムを散布するなどの手法で、容易に世界規模でのネットワーク障害を誘発させることができる。

【0017】

ネットワークを副次的に利用する、水道、ガス、電気などのライフライン、医療、軍事など、社会性を帯びたシステムが、このような攻撃を受ける可能性がある。

【0018】

例えばインターネット接続可能な携帯電話や、TVなどの装置にメール機能やWebブラウザ機能等を搭載する場合、受信パケットは、通常何階層ものプロトコルスタックモジュールで処理され、さらにアプリケーションで処理され、表示用の複数段の表示制御ソフトウェアの処理などを経て表示される。

【0019】

このように、何度もデータ解析処理が行われるが、このうち1つでも異常データに対する処理に不完全なものがあれば、セキュリティホールとなり得る。

【0020】

正常なデータと異なり、異常なデータのパターンは、ほぼ無限に組み合わせ可能であり、加えてタイミングやトラフィックの混雑によって、ネットワークの挙動が異なる場合があるため、完全な検証は極めて困難である。

【0021】

さらに開発規模が増大すると、人為的なバグ、すなわち悪意を持ったプログラマが開発に参入し、意図的にセキュリティホールを作成することも考慮に入れなければならない。

【0022】

実際、セキュリティの必要なソフトウェア開発の下請け業者に、テロ組織の要員が含まれていた事例も存在する。

【0023】

このような場合、セキュリティホールは、単にバッファオーバーフローのチェック機能の漏れや、チェック値の誤りを混入させるだけで良いから、レビューや検証による抽出で発見できない可能性も高く、発見できたとしても悪意によるものかどうか判定困難である。

【0024】

従って、プログラムソースコードを記述する際にプログラマが意識しなくとも、バグが顕在化したときに復帰できる機構をシステムに備える必要がある。

【0025】

(従来の技術1)

プロセッサ上で実行されるプログラム障害発生時に復帰する手法として、プログラムの

正常動作中、定期的にチェックポイントを通過させ、状態を保存しておき、プログラム障害が検出された場合、復帰処理実施後、最後に通過したチェックポイントにロールバックして処理を再開する方法がある（例えば特許文献1参照）。

【0026】

このエラー復帰方法は、C言語で書かれたプログラムに対しても適用可能である。この方法を、ある程度以上の精度を持つエラー検出手段と併せて適用すれば、プログラムバグが運用段階で顕在化した場合でも、システムをダウンさせずに処理続行ができるという利点がある。

【0027】

（従来の技術2）

また、C言語におけるメモリアクセス違反検出手法として、コンパイルの段階でポインタにアクセス可能な範囲情報を保持させ、アクセス毎にサブルーチンを呼び出して範囲チェックを行うようにソースコードを加工することで、既存C言語プログラムに、アクセス違反検出機構を埋め込む方法がある（例えば特許文献2参照）。

【特許文献1】特表平9-509515号公報

【特許文献2】特開平7-225703号公報

【発明の開示】

【発明が解決しようとする課題】

【0028】

（課題1）

従来の技術1の構成では、プログラムの構成を、定期的にチェックポイントを通過するように設定しなければならないため、プログラムの構造によっては、採用が困難な場合がある。

【0029】

（課題2）

また、従来の技術1で述べる構成では、最後に通過したチェックポイントから障害発生までの間の処理が2回実施されてしまう。

【0030】

従って、データを扱うだけのプログラムであれば、復帰処理過程である程度整合性を取ることができるが、通信制御などプログラムでは、同一のパケットを2度送出してしまったり、障害時に受信中だったパケットを損失してしまうなど副作用が発生するおそれがある。

【0031】

そこで本発明は、障害復帰機能を強化した情報処理装置を提供することを目的とする。

【課題を解決するための手段】

【0032】

請求項1記載の情報処理方法は、メモリアクセス違反を検出しながら通常処理を行うステップと、アクセス違反が検出されたら、アドレスに関連付けられた範囲情報と障害復帰情報とに基づいて復帰が可能か否かを判定するステップと、復帰が可能なら障害復帰を行うステップとを含む。

【0033】

この構成において、範囲情報と障害復帰情報とを、アドレスに関連付けることにより、障害復帰情報を用いて、障害個所のメモリ領域のデータタイプを特定できる。

【0034】

請求項2記載の情報処理方法では、復帰が可能でないなら停止処理を行うステップをさらに含む。

【0035】

この構成により、障害とは無関係なデータの破壊や、システムダウン等の危険を回避できる。

【0036】



請求項 3 記載の情報処理方法では、障害復帰情報は、固定サイズ属性と可変サイズ属性とを有する。

【0037】

この構成により、一定サイズ内に入っていない変数あるいはその変数を指すポインタと、サイズの変更が許される変数あるいはその変数を指すポインタとを、異なる態様で取り扱うことができる。

【0038】

請求項 4 記載の情報処理方法では、可変サイズ属性は、上方伸長属性と下方伸長属性とを有する。

【0039】

この構成により、サイズの変更が許される方向（アドレスにおける方向）が異なる変数あるいはその変数を指すポインタを、方向により異なる態様で取り扱うことができる。

【0040】

請求項 5 記載の情報処理方法では、停止処理を行うステップにおいて、保存可能なデータがあれば、このデータを保存する。

【0041】

この構成により、障害発生後、処理を再開あるいは検討する際に、保存したデータを利用できる。

【0042】

請求項 6 記載の情報処理方法では、障害復帰を行うステップにおいて、読み／書きのアクセス種別を判定し、判定結果に基づいて異なる内容の障害復帰を行う。

【0043】

この構成により、アクセス種別に応じて、異なる態様で障害復帰できる。

【0044】

請求項 7 記載の情報処理方法では、アクセス種別が「書き」であるとき、そのまま障害復帰し、アクセス種別が「読み」であるとき読み出し先に所定値を格納して障害復帰する。

【0045】

この構成により、アクセス種別が「書き」であるときは危険なメモリアクセスを回避し、「読み」であるときは読み出し側の不具合（例えば、暴走等）を回避できる。

【0046】

請求項 8 記載の情報処理方法では、判定するステップにおいて、上方伸長属性があり、かつ、アクセス違反が下方である場合、復帰は可能でないと判定する。

【0047】

請求項 9 記載の情報処理方法では、判定するステップにおいて、下方伸長属性があり、かつ、アクセス違反が上方である場合、復帰は可能でないと判定する。

【0048】

請求項 10 記載の情報処理方法では、判定するステップにおいて、固定サイズ属性がある場合、復帰は可能でないと判定する。

【0049】

これらの構成により、サイズ変更が不適な場合、復帰を取り止めて危険を回避できる。

【0050】

請求項 11 記載の情報処理方法では、障害復帰を行うステップにおいて、アクセス違反した領域とは別の領域を確保し、確保した別の領域にアクセスする。

【0051】

この構成により、別の領域へアクセスし、危険を回避できる。

【0052】

請求項 12 記載の情報処理方法では、障害復帰情報は、ターミネータ属性を有し、ターミネータ属性がデータの終端にターミネート値を有することを示す場合、障害復帰を行うステップにおいて、データの末尾にターミネート値を付加して障害復帰する。

## 【0053】

この構成により、ターミネート値に依存するプロセス等の安全性を向上できる。

## 【発明の効果】

## 【0054】

本発明によれば、情報処理装置の障害復帰機能を強化でき、動作安定性を向上できる。

## 【発明を実施するための最良の形態】

## 【0055】

以下図面を参照しながら、本発明の実施の形態を説明する。まず具体的な構成の説明に先立ち、本発明らによる考察を記述する。

## 【0056】

さて、従来の技術1において、チェックポイントまで戻らずに、障害復旧処理後に直接障害発生個所に戻ることが可能となれば、チェックポイントの設定が不要となり課題1が解決し、また、同じ部分を2度実行しないため課題2も解決する。

## 【0057】

そこで、C言語で記述されたプログラムの運用段階における障害において、障害復旧処理後に直接障害発生個所に戻るシステムが考えられる。

## 【0058】

障害復旧処理後に直接障害発生個所に戻る為には、まず、エラー検出の段階で次の条件がそろっている必要がある。

(条件1) 障害個所以外のデータを破壊する前に障害を検出できること

(条件2) 障害検出後、障害個所のメモリ領域が一意に特定できること

(条件3) 障害検出後、障害個所のメモリ領域の、C言語の意味でのデータタイプが特定できること

(条件4) 障害検出後、障害を検出した個所、またはその直後から処理再開できること

(条件1)、(条件2)、(条件4)を満たすエラー検出手段として従来の技術2の方法がある。

## 【0059】

この方法では、専用のコンパイラを使用することが前提となる。

## 【0060】

しかしながら、組み込み機器では、出荷前にROMに書き込まれたプログラム以外を実行することは無く、汎用的なアプリケーションソフトウェアの実行を想定したサーバー機やパーソナルコンピュータとは前提条件が異なる。

## 【0061】

組み込みシステムでは、製造者が利用するコンパイラ以外のコンパイラでコンパイルされたコードがROMに配置されることは無い。

## 【0062】

従って、完成したC言語ソースコードをROMに焼きこむためにコンパイルする段階で、障害復帰機能を組み込む事に問題は無い。

## 【0063】

しかしながら、従来の技術2は、主にプログラム運用前の検証工程におけるバグの早期発見を目的としており、そのまま運用時に利用することも可能であるが、以下の課題が問題になる場合もある。

(課題a) チェック処理がソフトウェア実行であるため処理量が著しく増大する

(課題b) チェック区間で割り込みが発生し、ポイントが変更されると整合性が取れなくなるため、割り込み禁止でチェックせねばならず、最大割り込み禁止時間が増大し、組み込みシステムではシステム破綻を引き起こす場合がある。

## 【0064】

しかしながら、(課題a)、(課題b)は、次のようにすれば解決可能である。

## 【0065】

まず、従来の技術2と同様に、コンパイラやライブラリに機能追加を施してポイントに

対して、ポインタの指すデータに対してアドレスだけではなく割り当てられたメモリ範囲の情報を付加する。簡略化するには、ポインタ型そのものを型拡張して、アドレスと範囲情報の両者を保持できるデータ型とすれば良い。

【0066】

この際、ポインタ型のサイズが増加するが、ANSI-C仕様においてポインタのデータ型を限定していないため、C言語の仕様を満たしたまま（C言語の仕様に違反せずに）実施可能である。

【0067】

この構成にあわせて、図1に示すようなプロセッサを用意すると良い。

【0068】

図1は、本発明の検討例におけるプロセッサ装置のブロック図である。

【0069】

図1に示すように、このプロセッサ装置10は、アドレスバス5及びデータバス6を介して、メモリ7やI/O装置8等に接続される。そして、プロセッサ装置10は、次の構成要素を備える。

【0070】

命令処理部1は、データバス6を経由して、メモリ7上のプログラムを読み込んで逐次実行したり、I/O装置8と入出力を行う。

【0071】

ここで、本検討例では、命令処理部1と、アドレスバス5との間に、MMU（メモリマネジメントユニット）4を設け、命令処理部1とMMU4とは、論理アドレスを用いて入出力をし、MMU4は、論理アドレス／物理アドレスの変換を行うことにしているが、例えば、MMU4を省略し、命令処理部1が、物理アドレスを入出力するようしても差し支えない。

【0072】

さて、本検討例のプロセッサ装置10には、通常のプロセッサ装置の要素に加えて、次のレジスタ（通常、汎用レジスタで構成すればよい。）が設けられている。

【0073】

まず、アドレスレジスタ2は、図1のプロセッサでは、アドレスレジスタから拡張したものであって、アドレスレジスタ2は、通常のアドレスレジスタが有するアドレスを保持する領域に加え、このアドレスに関する範囲情報を格納する領域を備えている。

【0074】

また、命令処理部1は、アドレスレジスタ2に対し、拡張されたポインタ型を1命令でロードストア可能な命令を処理できるものとする。

【0075】

1命令でロードストアするため、アドレスと範囲情報とが割り込みによって分断されることなくアトミックに操作可能であり、（課題b）を解決できる。

【0076】

図1において、アクセス違反検出部3は、比較器から構成される。アクセス違反検出部3は、命令処理部1が、アドレスレジスタ2に係るポインタを利用して、メモリ7をアクセスする際、命令処理部1の出力（本例では、論理アドレス値）と、アドレスレジスタ2の範囲情報とを入力し、これらの比較演算を行い、メモリ7の範囲違反があるときに、例外信号を命令処理部1へ出力する。

【0077】

このメモリ7のアクセスは、直接アドレッシングでも良いし、インデックス値の加減算を行う間接アドレッシングでも良い。

【0078】

このプロセッサアーキテクチャにより、範囲情報が不可分に関連付けられたポインタを、範囲チェックをしながら、1機械語命令により、高速かつアトミックに、扱うことが可能となる。即ち、ソフトウェア処理の増大なしに、アクセス違反を検出でき、（課題b）

を解決できる。

【0079】

しかも、OSなどプロセッサの特権モードで動作するプログラムの介在無しに、プログラム自身で、例外を発生させ、不適切なメモリアクセスをトラップできる。

【0080】

また、プログラムカウンタ9には、命令処理部1によって読み書きされ、かつ、プログラムにおける実行アドレス値が格納される。命令処理部1は、プログラムカウンタ9に新たな実行アドレス値を格納する処理を、1機械語命令において実行する。

【0081】

この処理は、プログラムカウンタ9の実行アドレス値を、インクリメントする場合や、JUMPあるいはCALL命令のような、分岐命令を実行する際に、行われる。

【0082】

次に、図2を参照しながら、このような構成におけるエラー検出の過程を説明する。

【0083】

図2(a)のソースコードでは、関数mainが、int型変数を3個分しかない配列aへのポインタを関数fooに渡し、関数fooがバッファをオーバーして4バイトのデータを書き込もうとする。

【0084】

まずコンパイラは、図2(a)のソースコードの3行目をコンパイルする時に、配列aのサイズが「3」であることを認識する。次に、コンパイラは、5行目において、配列aを指すポインタを生成して関数fooにパラメータとして渡すコードを生成するが、コンパイラは、配列aのアドレスだけでなく、配列aの割り当てられるメモリ範囲を示す範囲情報(上限アドレス、下限アドレス)を含んだポインタを生成する。

【0085】

コンパイル済みのコードが実行されると、関数fooのforループに対応するコードの実行時にループの4回目(ソースコードの14行目、i=4のとき)において、図2(d)に示すように、ポインタPは配列aの範囲を超えてアクセスしようとする。この際、ポインタ情報に範囲情報が含まれており、かつ、アドレスレジスタ2に、この範囲情報が格納されているから、アクセス違反検出部3がアクセス違反を検出し、例外信号を命令処理部1へ出力する。これにより、範囲を超えたアクセスを検出することが可能となる。

【0086】

以上の構成により、(条件1)、(条件2)、(条件4)は満たせるので、本発明では残る(条件3)を満たすために、コンパイラを利用してポインタ型に、障害復帰情報をさらに付加する。次に、この付加の原理を説明する。

【0087】

例として図3(a)及び図3(b)に示すソースファイルと、図3(c)に示す障害復帰情報ファイルを考える。

【0088】

図3(a)のソースファイルの8行目において、関数mainは、図3(b)のように定義される関数input\_strを呼出して、配列bufに文字列を取得する。そして、関数mainは、10行目~11行目において配列bufの文字列長をカウントし、13行目にて結果を標準出力に出力するものである。

【0089】

コンパイルの段階で、ソースコードとは別に、図3(c)に示すような障害復帰情報ファイルを用意し、障害復帰情報ファイルにおいてオブジェクト名、データ型、変数名などを指定し、ソースコード上のデータに対して障害復帰用の属性を付加する。なお、ここでのコンパイルとは、ソースから実行コードを出力する一連の過程を指しており、プリプロセス、リンクなども含んで広い意味でのコンパイルを指している。ここで、図3(c)の障害復帰情報ファイルは、元のソースコードそのものとは異なる存在であり、コンパイラにより評価されるから、コンパイラ擬似命令やマクロに似た存在であるということができ

る。

#### 【0090】

図3(c)の障害情報ファイルでは、main.c(図3(a))内のchar型の配列に対して、上方伸長属性(UPPER)を付加している。本形態では他に、下方伸長属性(LOWER)と固定サイズ(FIXED)とを取扱う。

#### 【0091】

図4(a)は関数mainから関数input\_strを呼び出した際のスタックメモリの内容を示し、図4(b)は、引数として渡されるポインタのデータ構造を示す。

#### 【0092】

関数呼び出しに際し、配列bufを指すポインタ(図4(b))が生成されるが、生成されるポインタ型には配列bufのアドレスのみならず、範囲情報(アドレス上限、アドレス下限)と障害復帰情報(UPPER)が含まれている。

#### 【0093】

char型の配列bufは、関数mainによりスタックメモリに図4(a)に示すように確保される変数であるが、図4(b)のポインタで範囲情報が渡されるために、関数input\_strは、メモリアクセス範囲違反を適切に監視できる。

#### 【0094】

ここで、関数input\_strにて、設計想定外のデータとして32バイト以上のデータが入力された場合、図3(b)のソースコード5行目のwhileループは32回以上実施され、図3(b)のソースコード6行目のアクセスでアクセス違反が発生し、アクセス違反検出部3は、命令処理部1に例外信号を出力する。

#### 【0095】

命令処理部1で実行される例外処理ハンドラを用意し、アクセス違反例外のリターンアドレスを調べれば、エラー発生個所の命令が判別できる。

#### 【0096】

命令を解析すれば、アクセスに用いられた操作が、書き込みか読み出しのいずれであるか、及び、この命令時のプログラムアドレスを特定できる。さらにアドレスレジスタ2から、違反時にアクセスしようとしたアドレス、範囲情報、及び障害復帰情報を取得可能である。

#### 【0097】

次に障害復帰情報をチェックする。図4(b)の例では、障害復帰情報は、上方伸長属性(UPPER)であるから、アドレス上方に伸びる可変長のデータを格納するデータ型であると判断でき、違反時にアクセスしようとしたアドレスが範囲情報の上側かどうかをチェックすれば良い。

#### 【0098】

この例では、上側にアクセス違反を起こしているため、バッファオーバーランが発生していると判断できる。

#### 【0099】

また、この例では、アクセスが「書き込み」であるから、バッファオーバーラン用の障害復帰処理として、アクセスエラーの発生した次のアドレスにリターンし、書き込みを無効化すると良い。

#### 【0100】

上記操作によれば、図4(a)に示すスタックメモリのbuf領域以外を一切破壊することなく、関数input\_strを終了させ得る。

#### 【0101】

さて、図3(a)のソースコード10行目において、関数mainは、whileループを実行し、NUL文字(ターミネート値の例)である「0」の検索を行う。

#### 【0102】

関数input\_strで読み出したデータの前半32バイトにNUL文字が1つも含まれていない場合、このwhileループは32回以上実行され、配列bufの領域を越

えてアクセスが発生することになり、アクセス違反が発生する。

【0103】

この場合、書き込み時と同様に例外処理を実行すれば良い。この際、書き込み時と異なるのは、読み出しアクセスである点だけである。

【0104】

読み出しアクセスの場合、障害復帰処理は、次のようにすれば良い。即ち、例外が発生した読み出しアクセスの結果に代えて、読み出し結果にターミネート値「0」を格納し、読み出しアクセス命令の次の命令にリターンする。

【0105】

こうすれば、関数mainのwhileループの終了条件が満たされるため、図4(a)に示すスタックメモリのbuf領域以外を一切破壊せずに、whileループを終了させ得る。

【0106】

本考察によれば、従来技術では、システムがクラッキングしてしまう、もしくはエラー検出しても強制終了しかできなかった攻撃又はバグに対して、プログラムの障害復帰を行うことができる。

【0107】

さらに、ソースコード自身にバッファオーバーフローを許すバグが意図的に埋め込まれている場合であっても、プログラムの障害復帰を行える。

【0108】

また、ここまでの例は、ポインタに障害復帰情報をもたせている。即ち、図5(a)のソースコードの7行目でパラメータ渡しが実施されるとき、図5(b)に示すようにポインタ値が障害復帰情報を保持するようにしている。

【0109】

これにかえて、図6に示すように、図5(a)のソースコードの1行目で宣言される配列a自身に、その範囲に隣接して障害復帰情報を持たせるようにしてもよい。

【0110】

このようにしても、アクセス違反が発生した場合、範囲情報が取得できるから、範囲情報に隣接するアドレスを読み出せば、図5(b)に示す場合と同様に、障害復帰情報を取得可能である。

【0111】

図5(b)によると、ポインタは、しばしばレジスタ経由でパラメータ渡しに利用されるから、パラメータ受け渡しの処理負荷が増える。

【0112】

一方、図6によると、ポインタアクセスする可能性のある全ての変数について障害復帰情報をつける必要があるから、メモリ使用量が増えるが、ポインタ渡しの処理量が削減できるメリットがある。以上で、本考察を終える。

【0113】

以上の考察をふまえ、以下本発明の骨子を説明する。

【0114】

(発明の骨子1)

本骨子は、コンパイル装置に関する。以上の考察を具体化すると、コンパイル装置は、図8に示すようになる。

【0115】

即ち、コンパイル装置30は、ポインタを取扱えるプログラム言語（例えば、C/C++言語やPascal等）で記述されたソースコード21（例えば、図3(a)、(b)参照）を読み込み、これをコンパイルして実行コードを出力する。なお、ここでいう「コンパイル」の意味は、上記考察で述べたとおりである。

【0116】

また、コンパイル装置30は、ソースコード21をコンパイルする際に、障害復帰情報

ファイル 22 (例えば、図 3 (c)) を参照する。なお、障害復帰情報は、コンパイル装置 30 が認識できる形態となっていれば十分であり、必ずしもファイルの形式で提供されなくとも良い。

【0117】

次が、重要な点である。即ち、コンパイル装置 30 は、ソースコード 21 を単にコンパイルするだけでなく、次の (関連付け 1) または (関連付け 2) のいずれかを実施する。

(関連付け 1) : ポインタについて、ポインタのアドレス、範囲情報及び障害復帰情報を不可分に関連付ける (図 5 (b) 参照)。

(関連付け 2) : ポインタについてポインタのアドレス及び範囲情報を不可分に関連付け、かつ、このポインタで操作される変数に障害復帰情報を不可分に関連付ける (図 6 参照)。

【0118】

(発明の骨子 2)

本骨子は、情報処理装置に関する。さらに詳しくは、この情報処理装置は、図 7 に示すプロセッサを備え、このプロセッサは、発明の骨子 1 におけるコンパイル装置 30 が生成した実行コードを実行する。

【0119】

図 7 は、本発明の骨子 2 におけるプロセッサのブロック図である。

【0120】

プロセッサ 20 のアドレスレジスタ 12 は、アドレス情報、範囲情報、及び、障害復帰情報を格納可能である。

【0121】

命令処理部 1 は、メモリ 7 から読み出したプログラムコードを解釈し、実行するが、この際にメモリ 7 上のアドレス情報、範囲情報、及び、障害復帰情報がセットになったポインタ型をアドレスレジスタ 12 にロード/ストアする命令を実行可能である。

【0122】

また、このプロセッサ 20 は、アクセス違反検出部 3 を備えており、命令処理部 1 がアドレスレジスタ 12 のアドレス情報を利用したアドレッシング命令を利用した場合、アクセス違反検出部 3 は、命令処理部 1 が出力する論理アドレスとアドレスレジスタの範囲情報を比較し、アクセスする論理アドレスが範囲外であった場合、命令処理部 1 に例外信号を出力する。

【0123】

命令処理部 1 は、例外信号を受け取ると、例外処理用の実行番地にその実行を分岐する。

【0124】

このプロセッサ 20 により、割り込み処理などによってポインタのもつアドレス情報、範囲情報、及び、障害復帰情報の操作を分断され、不整合を起こすことなく、且つ、高速にエラー検出を実施することが可能となり、さらに例外処理時にアドレスレジスタ 12 から障害復帰情報を取得することが可能となる。

【0125】

このときの本情報処理装置を機能ブロック図で示すと、図 9 のようになる。

【0126】

さて、通常処理部 23 は、メモリ 7 に格納されるプログラムを読み込み、アドレスレジスタ 12、プログラムカウンタ 9 を更新しながら通常の処理を行う。

【0127】

障害復帰判定部 24 は、通常処理部 23 がアクセス違反検出部 3 から例外信号を入力すると呼び出され、アドレスレジスタ 12 内の障害復帰情報を参照し、障害復帰の可否を判定する。

【0128】

復帰処理部 25 は、障害復帰判定部 24 が障害復帰可とする際、障害復帰判定部 24 に

呼出され、障害の状況に合わせて障害復帰処理を行い、処理を通常処理部 23 へ戻す。このとき、通常処理部 23 は通常処理を継続し、情報処理装置は停止しない。

【0129】

停止処理部 26 は、障害復帰判定部 24 が障害復帰不可とする際、障害復帰判定部 24 に呼出され停止処理を行い、処理を通常処理部 23 へ戻す。このとき、通常処理部 23 は通常処理を継続せず、情報処理装置は停止する。

【0130】

ここで、図 9 における通常処理部 23、障害復帰判定部 24、復帰処理部 25 及び停止処理部 26 は、図 7 における命令処理部 1 がメモリ 7 内のプログラムを実行することにより実現される。

【0131】

(実施の形態 1)

図 10 は、本発明の実施の形態 1 における情報処理装置のフローチャートである。

【0132】

以下、図 9 及び図 10 を参照しながら、通常処理部 23 がアクセス違反検出部 3 から例外信号を入力した後の処理を説明する。

【0133】

まずステップ 1 において、障害復帰判定部 24 は、通常処理部 23 から例外発生時のプログラムカウンタ 9 の値を取得し、アクセス違反が発生したアドレスの命令を取得する。

【0134】

取得した命令は、メモリアクセスを実施する命令のはずであり、読み出しか、書き込みか、どのアドレスレジスタを利用したかの情報を一意に特定できる。

【0135】

アドレスレジスタ 12 には、ポインタ情報が格納されているため、障害復帰判定部 24 は、本来アクセスすべきデータの範囲情報と障害復帰情報とを取得できる。

【0136】

次にステップ 2 において、障害復帰判定部 24 は、取得した障害復帰情報をチェックする。ここでは障害復帰情報として、上方伸長属性 (UPPER) と、下方伸長属性 (LOWER) と、固定サイズ属性 (FIXED) を例示するが、それ以外の属性を利用し、データ属性にあった復帰手段を用意することも可能である。

【0137】

ステップ 2 において、上方伸長属性 (UPPER) と判断された場合、障害復帰判定部 24 は、ステップ 3 において、アクセス違反がデータ範囲の上方に対して行われたかどうかをチェックする。アドレス上方でなかった場合は、障害復帰判定部 24 は障害復帰不可と判定し、停止処理部 26 を呼出す。ステップ 6 にて、停止処理部 26 は保存可能なデータがあればメモリ 7 に保存し (ステップ 6)、通常処理部 23 システムを停止させる。

【0138】

アドレス上方であれば、障害復帰判定部 24 は、バッファオーバーフロー (障害復帰可) と判定し、復帰処理部 25 を呼出して復帰を試みる。即ち、復帰処理部 25 は、ステップ 8 にてアクセス種別をチェックし書き込みであれば無効化し、読み出しであればステップ 9 にて固定値としてターミネート値 (例えば「0」) を読み出す。その後通常処理部 23 は、障害箇所の次のステップへ復帰する (ステップ 10)。

【0139】

ステップ 2 において、下方伸長属性 (LOWER) と判断された場合、ステップ 4 において、障害復帰判定部 24 は、アクセス違反がデータ範囲の下方に対して行われたかどうかをチェックする。アドレス下方でなかった場合は障害復帰判定部 24 は障害復帰不可と判定し、復帰処理部 25 を呼出す。ステップ 6 にて、停止処理部 26 は保存可能なデータがあればメモリ 7 に保存し (ステップ 6)、通常処理部 23 は、システムを停止させる (ステップ 7)。

【0140】



アドレス下方であれば、障害復帰判定部 24 は、バッファオーバーフロー（障害復帰可）と判定し、復帰処理部 25 を呼出し復帰を試みる。即ち、復帰処理部 25 は、ステップ 8 にてアクセス種別をチェックし、書き込みであれば無効化し、読み出しであればステップ 9 にて固定値としてターミネート値（例えば「0」）を読み出す。その後通常処理部 23 は、障害箇所次のステップへ復帰する（ステップ 10）。

#### 【0141】

ステップ 2 において、上方伸長属性でも下方伸長属性でも無い（FIXED）と判断された場合、障害復帰判定部 24 は、障害復帰不可と判定し、復帰処理部 25 を呼出す。ステップ 6 にて、停止処理部 26 は保存可能なデータがあれば保存し（ステップ 6）、通常処理部 23 は、システムを停止させる（ステップ 7）。

#### 【0142】

（実施の形態 1 の効果）

以上のように、アクセス違反例外を処理することで、データの範囲を超えた書き込み処理において、無関係のデータを破壊することなく処理継続を可能にし、さらに、データの範囲を超えた読み出し処理においても、無関係のデータを読み出すことなく、安全にデータを読み出すことができる。

#### 【0143】

図 11 は、本発明の実施の形態 1 におけるコンパイル装置の機能ブロック図ある。

#### 【0144】

ソースコード 21 は、言語解析部 34 に入力され、言語解析部 34 はソースコード 21 を実行コードへ翻訳する。ここで、言語解析部 34 は、ポインタ生成において、変数へのアドレスと、範囲情報と、障害修復情報が入るべきコードを確保するが、格納する値は未定値のままである。

#### 【0145】

また、言語解析部 34 は、変数宣言部分を見つけると、変数に割り当てた領域のアドレスと範囲情報とを変数領域記憶部 35 に書き込んで保存する。

#### 【0146】

翻訳された実行コードは、コード生成部 36 に送られる。

#### 【0147】

障害復帰情報読み込み部 31 は、ソースコード 21 とは異なる障害復帰情報ファイル 22 から情報を読み出し、障害復帰情報記憶部 32 に格納する。

#### 【0148】

コード生成部 36 は、言語解析部 34 から出力される実行コードの中の、未定位置となっているポインタの内部の値について、変数アドレスと、範囲情報について変数領域記憶部 35 から読み出して格納する。

#### 【0149】

また、コード生成部 36 は、ポインタの指す変数の情報を検索部 33 に送る。検索部 33 では、入力された変数情報から、障害復帰情報記憶部 32 内を検索し、変数に対応する障害復帰情報をコード生成部 36 に出力する。コード生成部 36 は、検索部 33 から得られた検索結果としての障害復帰情報をコード中のポインタ内に書き込む。

#### 【0150】

これにより、コード生成部 36 は、変数アドレスと、範囲情報、障害復帰情報がポインタ部分に格納された実行コード 40 を出力する。

#### 【0151】

本コンパイル装置により、ソースコード 21 から障害復帰情報の埋め込まれた実行コードが生成されるため、生成された実行コードを情報機器にインストールすれば、この情報機器に障害復帰機能を持たせることができる。

#### 【0152】

図 12 は、本発明の実施の形態 1 におけるコンパイル装置のフローチャートである。

#### 【0153】

ステップ20において、言語解析部34は、ソースコード21の先頭から順次ソース解析を実施する。

【0154】

ステップ20において、変数宣言であった場合、ステップ21にて、言語解析部34は、変数用の領域を確保し、その情報を変数領域記憶部35に保存する。

【0155】

ステップ20において、変数からポインタを生成する命令であった場合、ステップ22にて、言語解析部34は、ポインタ値として、変数のアドレス、変数の範囲情報、障害復帰情報を含んだ値を生成するコードを生成する。

【0156】

ここでは、ポインタ内の値は決定されないため、言語解析部34は、値が未定のコードを生成する。

【0157】

ステップ20において、ポインタ演算であった場合、ステップ23にて、言語解析部34は、ポインタ内のアドレス値のみを演算し、範囲情報及び障害復帰情報はそのままコピーするコードを生成する。

【0158】

ステップ20において、上記のいずれでもなかった場合、ステップ24にて、言語解析部34は、通常のコンパイラと同様に言語に対応するコードを生成する。

【0159】

言語解析部34がここまでの手順をソースコード終端まで繰り返した後（ステップ25）、コード生成部36は、コード内の未定となっているポインタ情報のうち、アドレス情報と範囲情報を、変数領域記憶部35に保存されている変数用の領域の情報から決定し、格納する（ステップ26～28）。

【0160】

最終工程として、コード生成部36は、ポインタ情報内の障害復帰情報が未定となっているポインタを検索する。検索部33は、障害復帰情報が未定となっているポインタが指し示している変数の属性を、障害復帰情報記憶部32から検索し、一致する設定があれば、コード生成部36は、その値を格納し、検出されなければ固定の値、例えば固定サイズを示す属性（FIXED）を設定する。この操作は、未定値がなくなるまで繰り返される（ステップ26～30）。

【0161】

なお、ここでは、ソースコードから実行可能コードを生成するまでの一連の工程を「コンパイル」と呼んでおり、狭い意味でのコンパイルやリンクなどの工程に分割しても構わない。

【0162】

（実施の形態1の変形例）

図13は、図10のステップ8以降に（図9における復帰処理部25の処理内容）に変更を加えたものである。

【0163】

説明の重複を避けるため、ステップ8以降の処理のみ説明する。ステップ8にて、復帰処理部25は、アクセス種別をチェックする。書き込みであれば、復帰処理部25はアクセスすべきデータとは別の領域をメモリに確保し、書き込もうとしていたデータを確保した領域に書き込む（ステップ34～36）。

【0164】

読み出しであれば、ステップ31にて、復帰処理部25は、既に書き込もうとしていたデータをステップ35で確保した領域が無いかチェックし、存在すればステップ32にて復帰処理部25は、この領域の値を読み出し、存在しなければステップ33にて固定値としてターミネート値（例えば「0」）を読み出す。

【0165】

(変形例の効果)

上記のように、アクセス違反例外を処理することで、データの範囲を超えた書き込み処理において、自動的にデータサイズを拡張することが可能となる。

【0166】

入力サイズが見積もり値を超えた際、単に破棄するだけでは破綻してしまうようなデータに対する復帰が可能となる。

【0167】

(実施の形態2)

図14は、本発明の実施の形態2におけるコンパイル装置の機能ブロック図である。図14において、実施の形態1に係る図11と同様の構成要素には、同一符号を付すことにより、説明を省略する。

【0168】

さて、実施の形態1では、図5(b)に示すように、ポインタ値に障害復帰情報を持たせた(関連付け1)が、実施の形態2では図6に示すようにポインタで操作される変数に障害復帰情報を持たせる(関連付け2)。

【0169】

このため、実施の形態1におけるコード生成部36(図11参照)を、図14に示すように、第1コード生成部37と第2コード生成部38とに分けている。また、言語解析部34'の処理内容が変更されている。

【0170】

また、言語解析部34'はソースコード21において、変数宣言部分を見つけると、変数に割り当てた領域のアドレスと範囲情報とを変数領域記憶部35に書き込んで保存すると共に、変数領域に隣接する場所に障害復帰情報が入る容量を余分に確保する。これにより、図6に示す変数領域が確保される。

【0171】

また、第1コード生成部37は、言語解析部34'から出力される実行コードの中の、未定位置となっているポインタの内部の値について、変数アドレスと、範囲情報について変数領域記憶部35から読み出して格納する。

【0172】

そして、第2コード生成部38は、第1コード生成部37から出力された、変数アドレスと、範囲情報がポインタ部分に格納された実行コードを受け取り、変数情報を順次検索部33に送信し、検索部33から得られた検索結果としての障害復帰情報を変数情報に隣接する領域に格納する。

【0173】

他の点は、実施の形態1と同様である。

【0174】

図15は、本発明の実施の形態2におけるコンパイル装置のフローチャートである。

【0175】

ステップ40にて、言語解析部34'は、ソースコード21の先頭から順次ソース解析を実施する。

【0176】

ステップ40において、変数宣言であった場合、言語解析部34'は、変数用の領域と変数用の領域に隣接する障害復帰情報格納用の領域を確保し、その情報を保存する(ステップ41、42)。

【0177】

ステップ40において、変数からポインタを生成する命令であった場合、ステップ43にて、言語解析部34'は、ポインタ値として、変数のアドレス、変数の範囲情報、障害復帰情報を含んだ値を生成するコードを生成する。

【0178】

ここでは、ポインタ内の値は決定されないため、言語解析部34'は、値が未定のコー

ドを生成する。

【0179】

ステップ40において、ポインタ演算であった場合、ステップ44にて、言語解析部34'は、ポインタ内のアドレス値のみを演算し、範囲情報及び障害復帰情報はそのままコピーするコードを生成する。

【0180】

ステップ40において、上記のいずれでもなかった場合、ステップ45にて、言語解析部34'は従来のコンパイラと同様に言語に対応するコードを生成する。

【0181】

言語解析部34'がここまでの手順をソースコード終端まで繰り返した後（ステップ46）、ステップ47にて、第1コード生成部37は、コード内の未定となっているポインタ情報のうち、アドレス情報と範囲情報を、保存されている変数用の領域の情報から決定し、格納する。

【0182】

最終工程として、第2コード生成部38は、データ情報に付随する情報内の障害復帰情報が未定となっているポインタを検索する。検索部33は、障害復帰情報が未定となっているポインタが指し示している変数の属性を、障害復帰情報記憶部32から検索し、一致する設定があれば、第2コード生成部38は、その値を格納し、検出されなければ固定の値、例えば固定サイズを示す属性（FIXED）を設定する。この操作は、未定値がなくなるまで繰り返される（ステップ48～51）。

【0183】

なお、ここでは、ソースコードから実行可能コードを生成するまでの一連の工程を「コンパイル」と呼んでおり、狭い意味でのコンパイルやリンクなどの工程に分割しても構わない。

【0184】

（実施の形態3）

図16は、本発明の実施の形態3における、情報処理装置のフローチャートである。なお、実施の形態3の機能ブロック図は、実施の形態1と同じく図9である。以下、実施の形態1との相違点を中心に説明する。

【0185】

ステップ2において、障害復帰判定部24は、取得した障害復帰情報をチェックする。ここでは障害復帰情報として、上方伸長属性（UPPER）と、下方伸長属性（LOWER）と、固定サイズ属性（FIXED）を例示するが、それ以外の属性を利用し、データ属性にあった復帰手段を用意することも可能である。

【0186】

ステップ2において、上方伸長属性（UPPER）と判断された場合、実施の形態1と同様に、ステップ52にて、障害復帰判定部24は、上方伸長属性の復帰処理を実施する。更に、ステップ54にて、障害復帰判定部24は、データにターミネータ属性が付されている場合、ステップ55にてデータ内にターミネータデータが存在するかどうか検索し、ステップ56にてデータが存在しなければ、ステップ57にて、障害復帰判定部24は、データ末尾にターミネータデータ（例えば、「0」）を追加する。これにより、失われてしまったターミネータデータを復元でき、データの信頼性を向上できる。

【0187】

なお、ステップ53において、下方伸長属性（LOWER）と判断された場合、障害復帰判定部24は、実施の形態1と同様の復帰処理を実施し、上方伸長属性でも下方伸長属性でも無い（FIXED）と判断された場合も、障害復帰判定部24は、実施の形態1と同様の処理を実施する。

【0188】

（実施の形態3の効果）

例えばC言語において、文字列のようなデータは、その終端にNUL文字を付加するこ

とでデータの終端を表している。

【0189】

しかしながら、不正なデータなどによりデータ構造が破壊されると、データ内にNULL文字が存在しないことになり、本来、アクセスしてはならないアドレスのデータを書き換えてしまうなど、後のデータ操作で深刻な障害を引き起こしやすくなる。

【0190】

本形態によれば、障害発生時に文字列のようなデータ構造を正規の形式に強制的に正規化することが可能となり、副次的な障害発生を抑制できる。

【0191】

(実施の形態4)

図17は、本発明の実施の形態4におけるコンパイル装置のブロック図であり、図18は、その装置のフローチャートである。

【0192】

さて、実施の形態1～3では、ソースコード21に直接変更を加えずに、コード生成を行った。

【0193】

一方、本形態では、コンパイル装置の内部において、ソースコード21に情報を付加し中間コードを生成する。例えば、図3(a)、(b)に示すソースコードがあり、図3(c)に示す障害復帰情報ファイルがあるときに、図3(a)のソースコードに基づいて図19(a)の中間コードを生成し、図3(b)のソースコードに基づいて図19(b)の中間コードを生成する。なお、中間コードの生成自体は、例えば文字列の置換等、周知技術を用いて簡単に実装できる。

【0194】

図19(a)の中間コード5行目、6行目に、「UPPER」、「FIXED」というキーワードが挿入されており、同様に、図19(b)の中間コード3行目に「FIXED」というキーワードが挿入されている。

【0195】

図17において、障害復帰情報付加部39は、ソースコード21を入力し、ソースコード21に障害復帰情報を付加した中間コードを生成する。その他の点は、実施の形態1と同様である。

【0196】

ここで、実施の形態1～3及び本形態において、プログラマ自身がソースコード21に直接手を加えなくても、障害復帰情報を追加できる点に注目されたい。よくある状況であるが、既存のソースコードが多量存在し、これらに基づく実行コードを、本発明の障害復帰機能を持つバージョンへ一括して変更したいような場合、この点は極めて有利になる。

【0197】

次に、図18を参照しながら、本形態のコンパイル装置の動作を説明する。まずステップ70にて、障害復帰情報読み込み部31は障害復帰情報ファイル22を読み込み、障害復帰情報記憶部32に障害復帰情報をセットする。

【0198】

次に、ステップ71にて、障害復帰情報付加部39は、ソースコード21を1ステップ(例えば、1行等)読み込み、ステップ72にて、このステップに変数宣言があるかどうかチェックする。なければ、ステップ75へ処理が移る。

【0199】

変数宣言があれば、ステップ73にて、障害復帰情報付加部39は検索部33に変数情報を出力し、この変数情報が示す変数の障害復帰情報を検索する。この情報が存在すれば、ステップ74にて、障害復帰情報付加部39は、変数宣言に障害復帰情報を付加し、ステップ75へ処理が移る。

【0200】

ステップ75において、障害復帰情報付加部39はこのステップについての中間コード

をテンポラリファイルとして外部（例えば、ハードディスク装置の一定領域等）へ出力する。ステップ76にて、障害復帰情報付加部39は、ソースコード21の終端に至っていないことを確認し、ソースコード21の終端に至るまで、ステップ71～76の処理を繰り返す。

【0201】

これにより、図19(a)、(b)に示すような中間コードが生成されていることになる。

【0202】

中間コードが生成された後は、実施の形態1において、「ソースコード21」とあるのを「中間コード」と読み替えた処理が行われる（ステップ77～83）。

【0203】

本形態によっても、実施の形態1と同様の効果がある。

【0204】

（実施の形態5）

図20は、本発明の実施の形態5における障害情報監視システムのブロック図である。

【0205】

情報処理装置50は、図7に示すプロセッサ20、アドレスバス5、データバス6、メモリ7及びI/O装置8を備える。メモリ7には、図9に示す機能を実装したプログラム（通常のアプリケーションコードの他、障害復帰コードを含むもの）がロードされている。

【0206】

I/O装置8は、伝送路55を介して情報センター60のI/O装置61と接続している。情報センター60は、I/O装置61の他、障害情報を記録する記憶装置62と、障害情報を表示する表示装置63とを有する。

【0207】

情報処理装置50において、アプリケーションコードの実行時に障害が発生すると、プロセッサ20は、メモリ7内の障害復帰コードを実行する。

【0208】

障害復帰コードが実行されると、障害内容の情報が、伝送路55を介して情報センター60に送信される。

【0209】

情報センター60では、障害内容が記録装置62に記録され、表示装置63が障害の内容を表示する。

【0210】

（実施の形態5の効果）

このようなシステム構成によって、例えば携帯電話のような各ユーザーに販売される情報処理装置の運用段階での障害情報を収集することが可能となる。

【0211】

従って、情報処理装置の次回出荷分のロットで障害個所を修正したり、他の開発において類似の障害を残さないように開発方法を改善するのに役立つことができる。

【0212】

（実施の形態6）

図21は、本発明の実施の形態6における障害情報監視システムのブロック図である。本形態は、実施の形態5に対し、情報センター60に、変更を加えたものである。

【0213】

情報センター60では、障害に対する復帰方法の情報を入力装置65から入力し、記憶装置62に蓄積しておくことが可能である。

【0214】

情報処理装置50において、アプリケーションの実行時に障害が発生すると、プロセッサ20は、障害復帰コードを実行する。これにより、障害内容の情報が、伝送路55を介

して情報センター 60 に送信され、情報センター 60 では、検索装置 64 が記憶装置 62 を検索し、今回の障害に対応する復帰方法の情報を送り返す。

【0215】

情報処理装置 50 では、送り返されてきた復帰方法の情報に従って復帰処理を行う。

【0216】

(実施の形態 6 の効果)

このようなシステム構成によって、例えば携帯電話のような各ユーザーに販売される情報処理装置の運用段階での障害復帰手段の変更が可能となる。

【0217】

事前に組み込んだ障害復帰機構のみで対応できないトラブルが発生した場合でも、情報センター側で対処方法を登録することで、トラブルを収拾することが可能となる。

【産業上の利用可能性】

【0218】

本発明に係る方法は、例えば、ポインタを扱える言語を用いたプログラム開発の分野等において広く好適に利用できる。

【図面の簡単な説明】

【0219】

【図 1】 本発明の考察におけるプロセッサ等のブロック図

【図 2】 (a) 本発明の考察におけるソースコードの例示図 (b) 本発明の考察におけるポインタアクセスの説明図 (c) 本発明の考察におけるポインタアクセスの説明図 (d) 本発明の考察におけるポインタアクセスの説明図

【図 3】 (a) 本発明の考察におけるソースコードの例示図 (b) 本発明の考察におけるソースコードの例示図 (c) 本発明の考察における障害復帰ファイルの例示図

【図 4】 (a) 本発明の考察におけるスタックメモリの説明図 (b) 本発明の考察におけるポインタの説明図

【図 5】 (a) 本発明の考察におけるソースコードの例示図 (b) 本発明の考察におけるポインタと変数の関係図

【図 6】 本発明の考察におけるポインタと変数の関係図

【図 7】 本発明の実施の形態 1 におけるプロセッサ等のブロック図

【図 8】 本発明の実施の形態 1 におけるコンパイル装置の概念図

【図 9】 本発明の実施の形態 1 における情報処理装置の機能ブロック図

【図 10】 本発明の実施の形態 1 における情報処理装置のフローチャート

【図 11】 本発明の実施の形態 1 におけるコンパイル装置の機能ブロック図

【図 12】 本発明の実施の形態 1 におけるコンパイル装置のフローチャート

【図 13】 本発明の実施の形態 1 (変形例) におけるコンパイル装置のフローチャート

【図 14】 本発明の実施の形態 2 におけるコンパイル装置の機能ブロック図

【図 15】 本発明の実施の形態 2 におけるコンパイル装置のフローチャート

【図 16】 本発明の実施の形態 3 における情報処理装置のフローチャート

【図 17】 本発明の実施の形態 4 におけるコンパイル装置の機能ブロック図

【図 18】 本発明の実施の形態 4 におけるコンパイル装置のフローチャート

【図 19】 (a) 本発明の実施の形態 4 におけるソースコードの例示図 (b) 本発明の実施の形態 4 におけるソースコードの例示図

【図 20】 本発明の実施の形態 5 における障害情報監視システムのブロック図

【図 21】 本発明の実施の形態 6 における障害情報監視システムのブロック図

【図 22】 (a) 従来のソースコードの例示図 (b) 従来のスタックメモリの説明図 (c) 攻撃用データの例示図

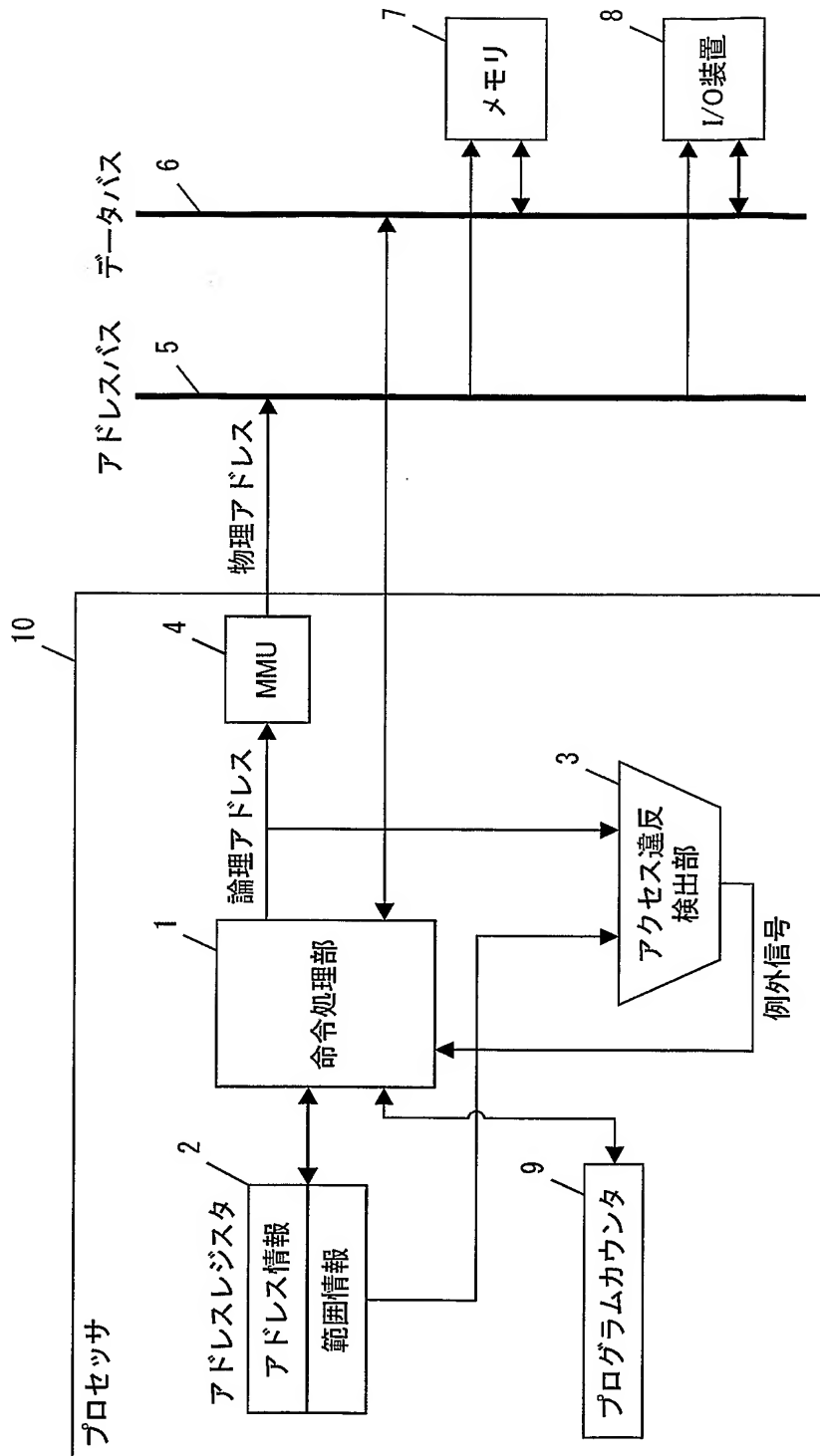
【符号の説明】

【0220】

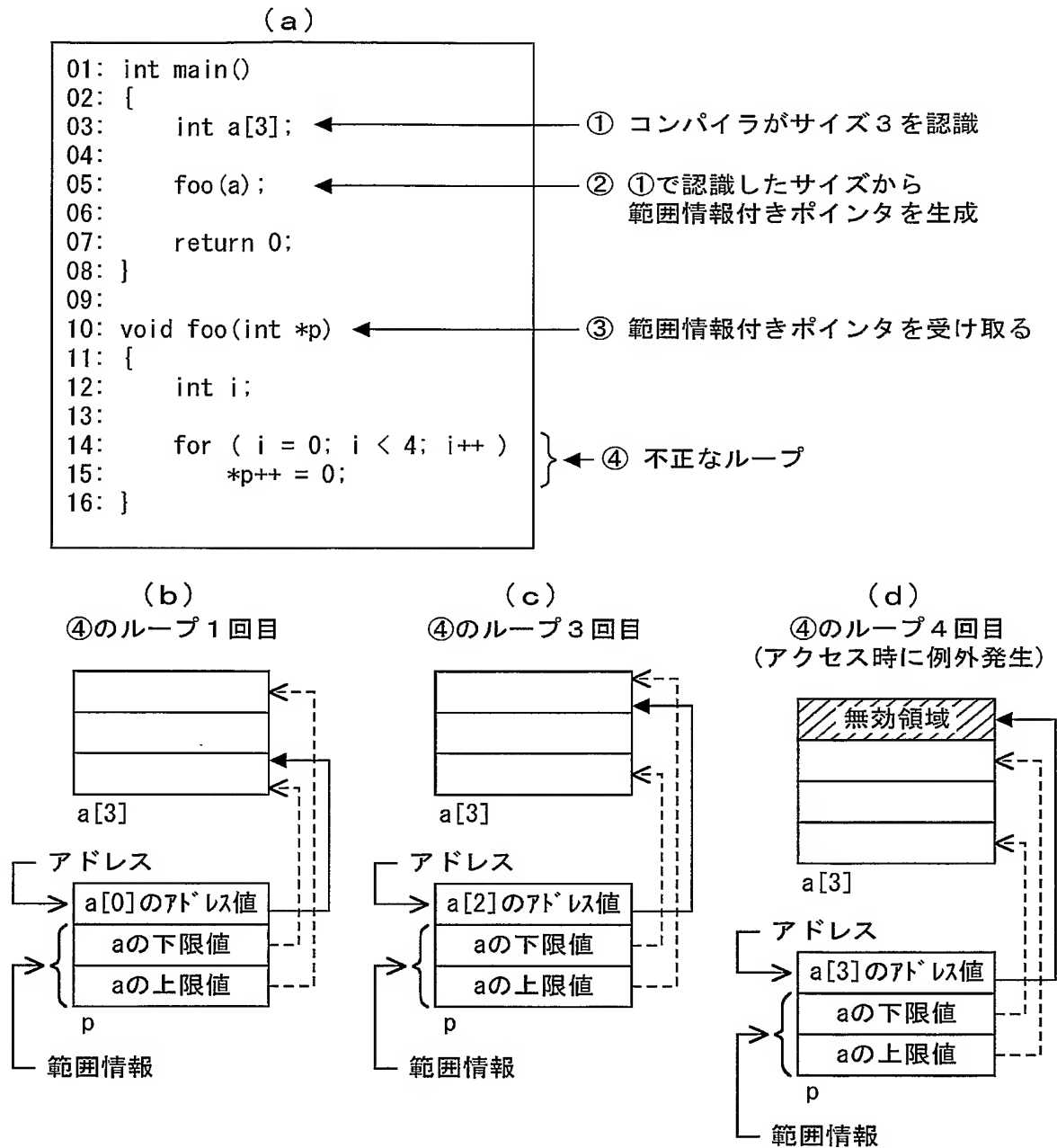
- 1 命令処理部
- 2 アドレスレジスタ
- 3 アクセス違反検出部
- 4 メモリマネージメントユニット
- 5 アドレスバス
- 6 データバス
- 7 メモリ
- 8 I/O装置
- 9 プログラムカウンタ
- 1 0 プロセッサ
- 1 2 アドレスレジスタ
- 2 0 プロセッサ
- 2 1 ソースコード
- 2 2 障害復帰情報ファイル
- 2 3 通常処理部
- 2 4 障害復帰判定部
- 2 5 復帰処理部
- 2 6 停止処理部
- 3 0 コンパイル装置
- 3 1 障害復帰情報読み込み部
- 3 2 障害復帰情報記憶部
- 3 3 検索部
- 3 4、3 4' 言語解析部
- 3 5 変数領域記憶部
- 3 6 コード生成部
- 3 7 第 1 コード生成部
- 3 8 第 2 コード生成部
- 3 9 障害復帰情報付加部
- 4 0 実行コード
- 5 0 情報処理装置
- 5 5 伝送路
- 6 0 情報センター
- 6 1 I/O装置
- 6 2 記憶装置
- 6 3 表示装置
- 6 4 検索装置
- 6 5 入力装置



【書類名】 図面  
【図 1】



【図 2】



【図 3】

(a)

```
01: extern void input_str(char *str);
02:
03: int main()
04: {
05:     char buf[32];
06:     int len = 0;
07:
08:     input_str(buf);
09:
10:     while ( buf[len] != 0)
11:         len++;
12:
13:     printf( "%d文字です¥n" , len);
14:
15:     return 0;
16: }
```

ソースコード(main.c)

(b)

```
01: void input_str(char *str)
02: {
03:     int c;
04:
05:     while((c = getchar()) != EOF)
06:         *str++ = c;
07: }
```

ソースコード(input\_int.c)

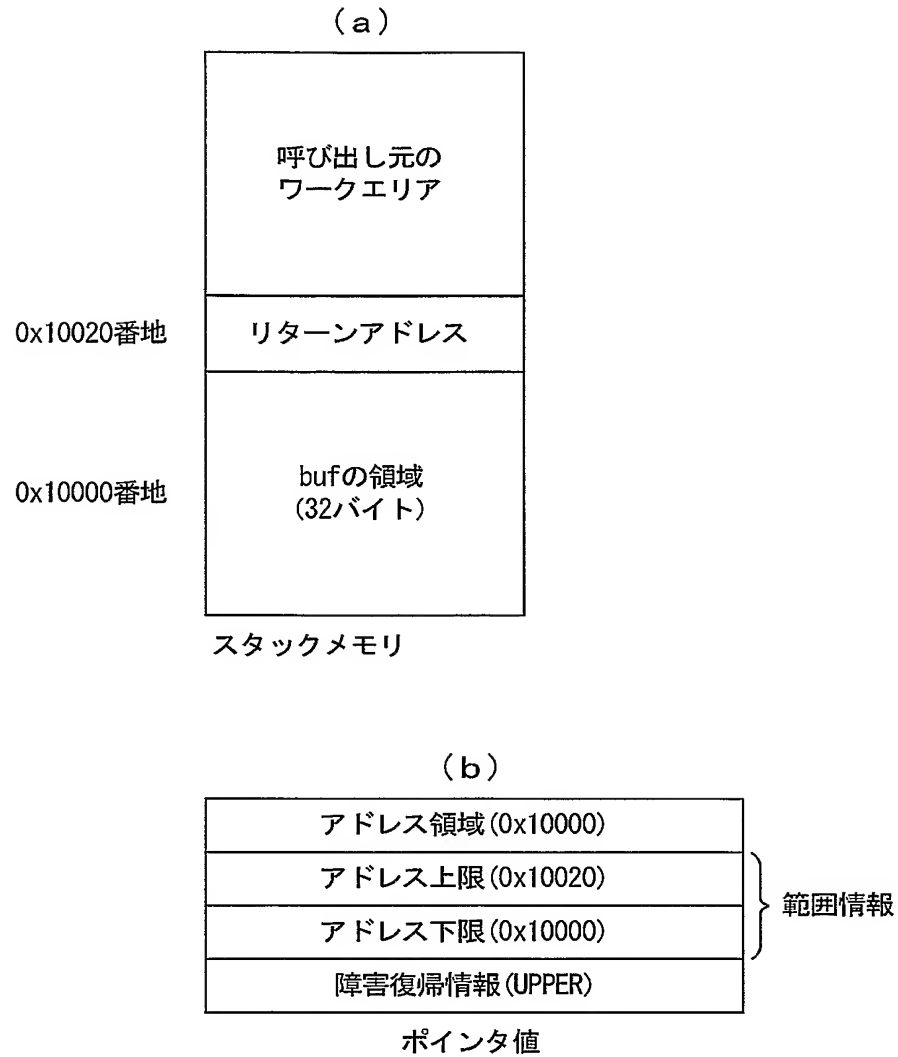
(c)

```
main.c:
    char[] UPPER      ;上方伸長
    *      FIXED      ;固定サイズ

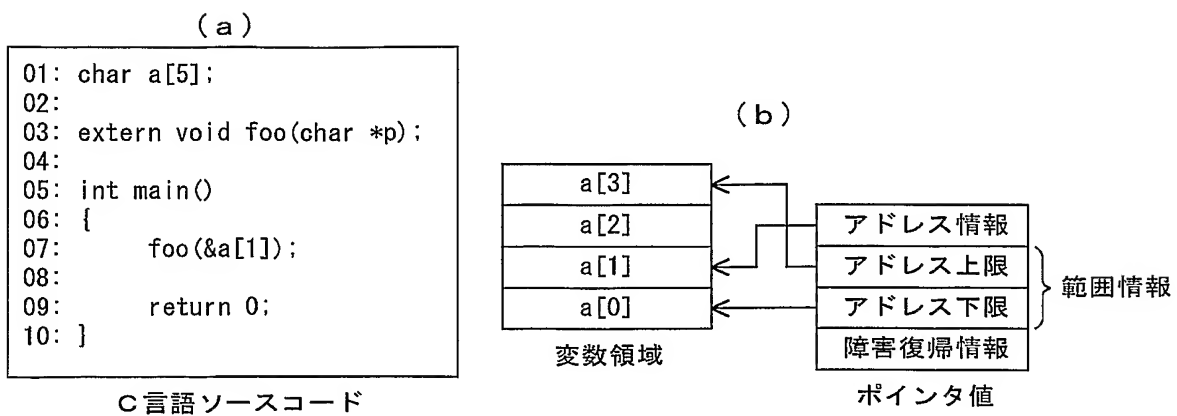
input_str.c:
    *      FIXED      ;固定サイズ
```

障害復帰情報ファイル

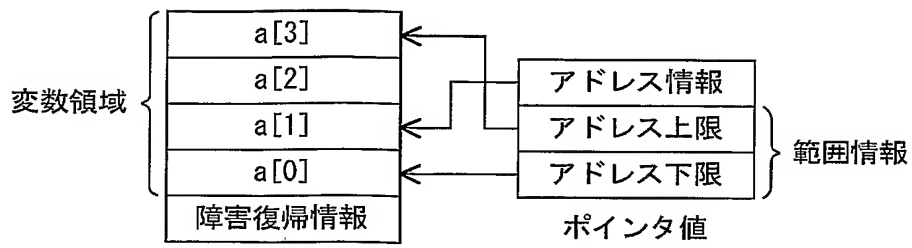
【図 4】



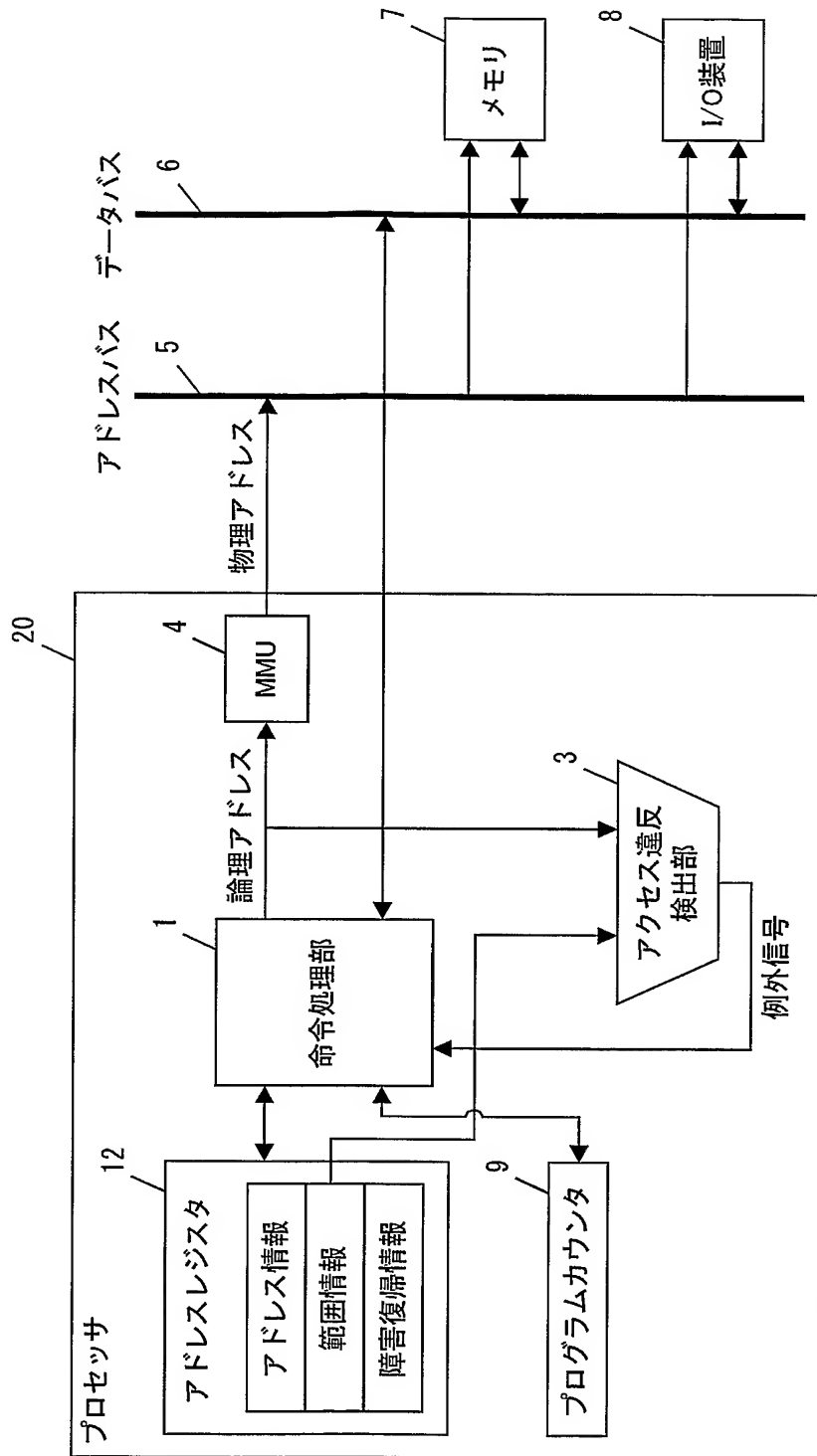
【図 5】



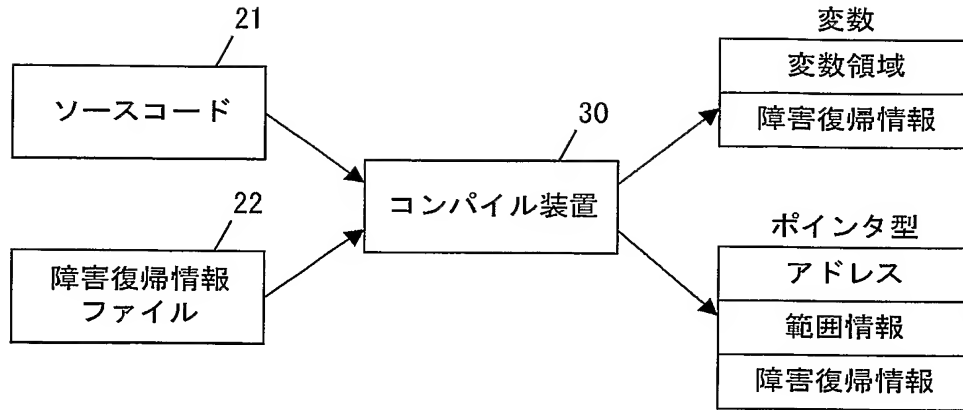
【図 6】



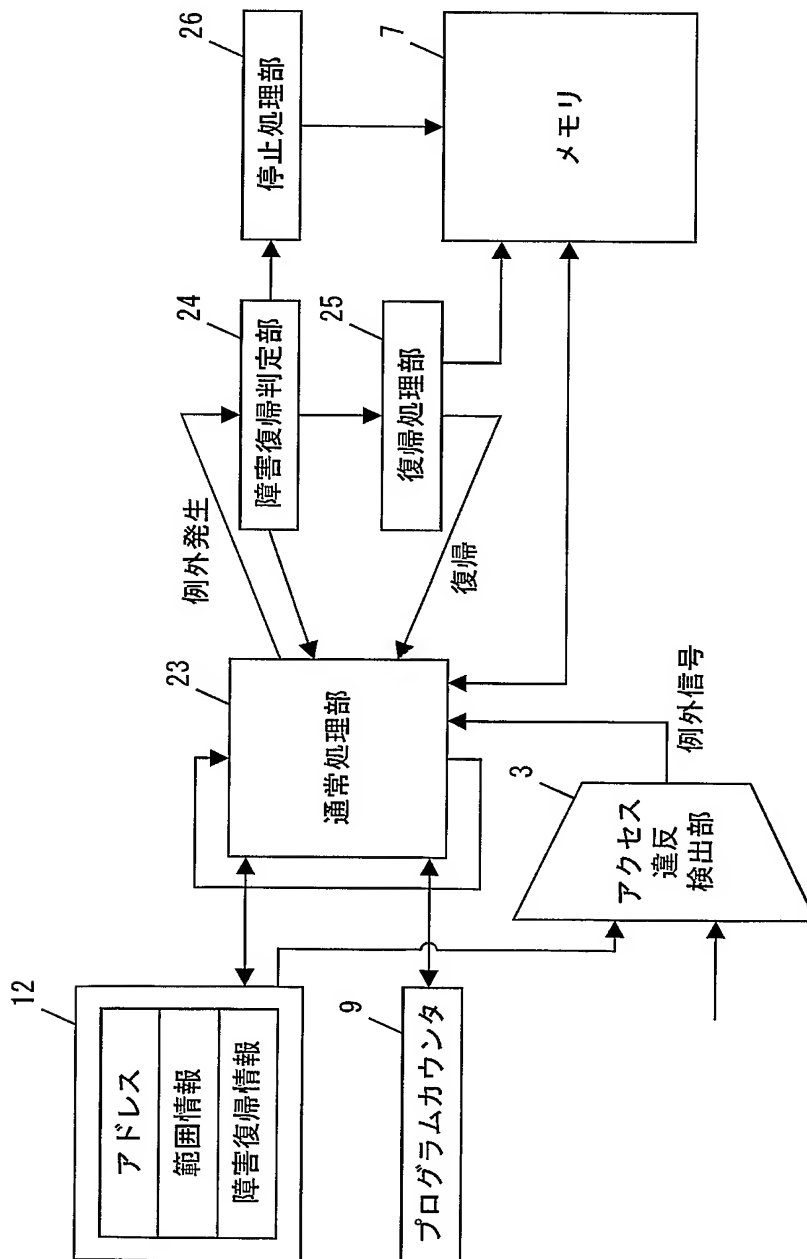
【図 7】



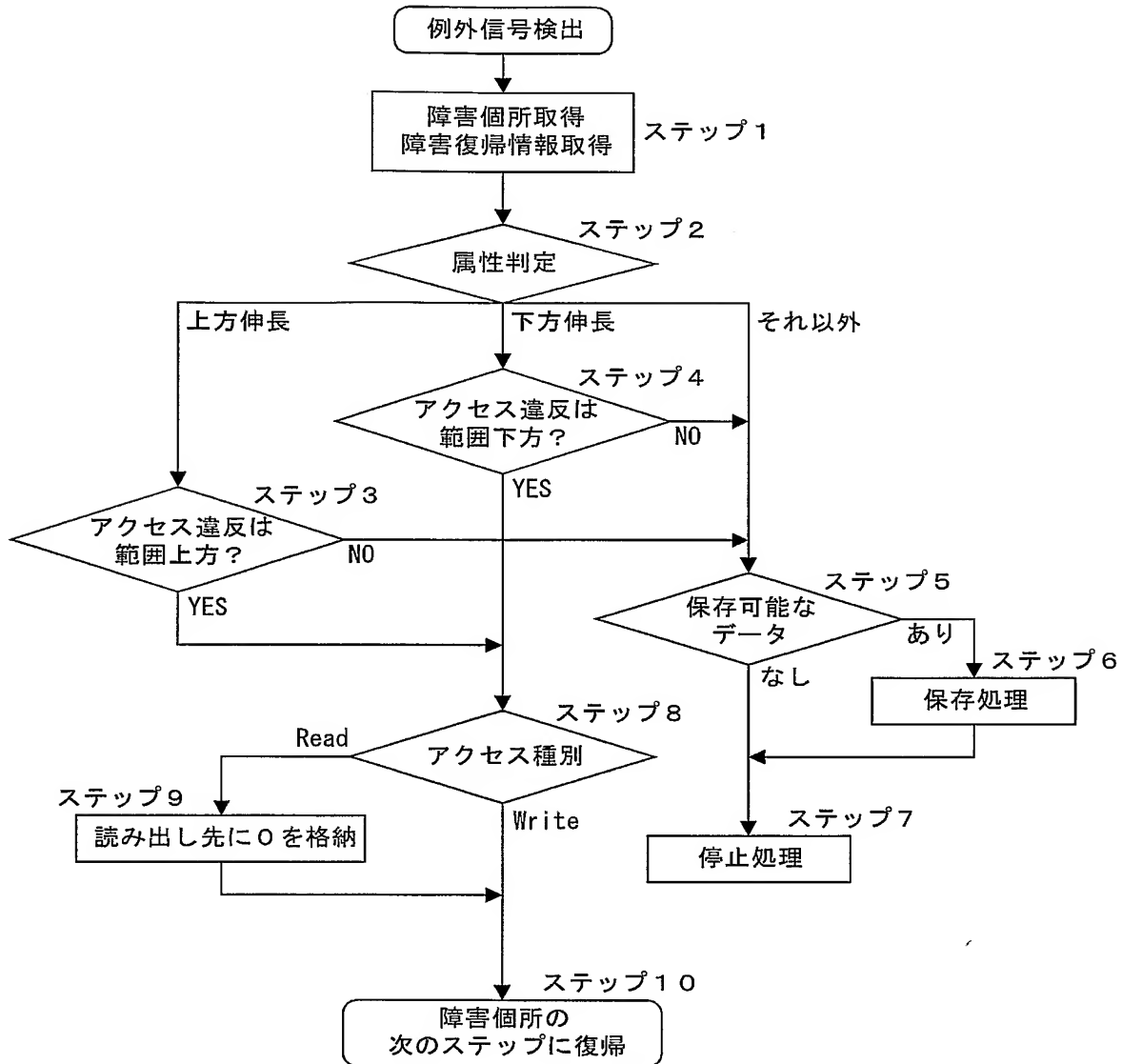
【図 8】



【図 9】

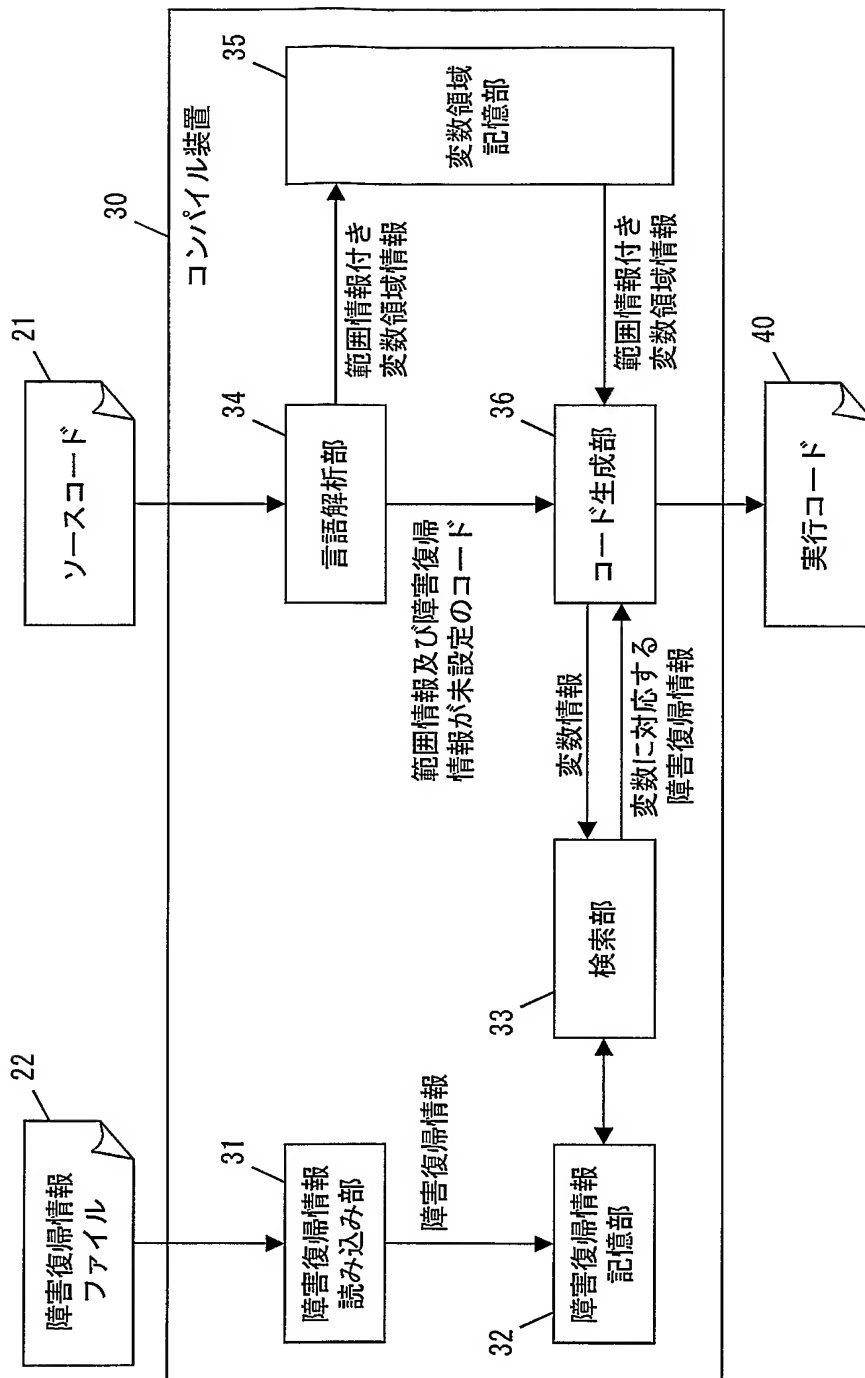


【図 10】

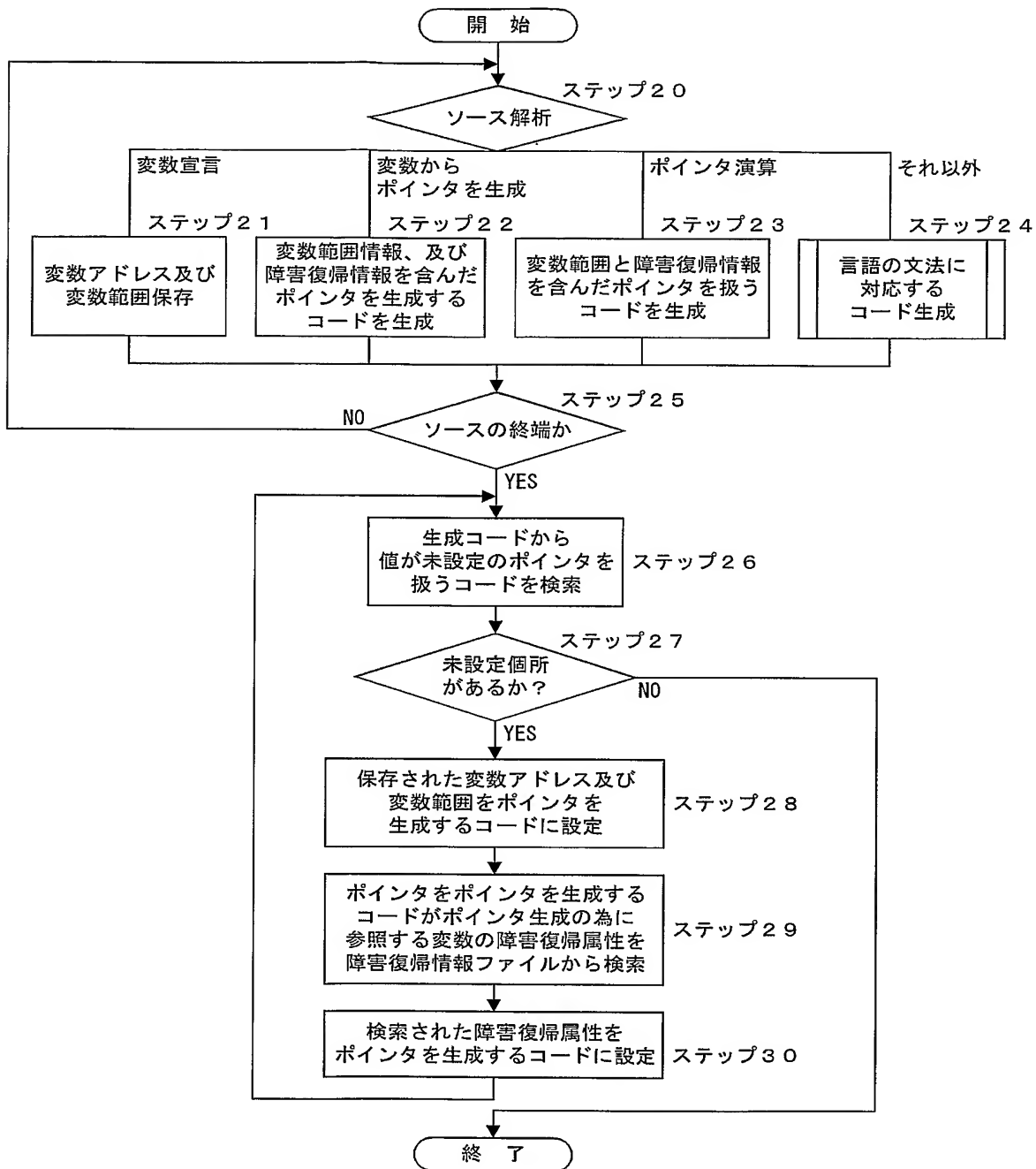




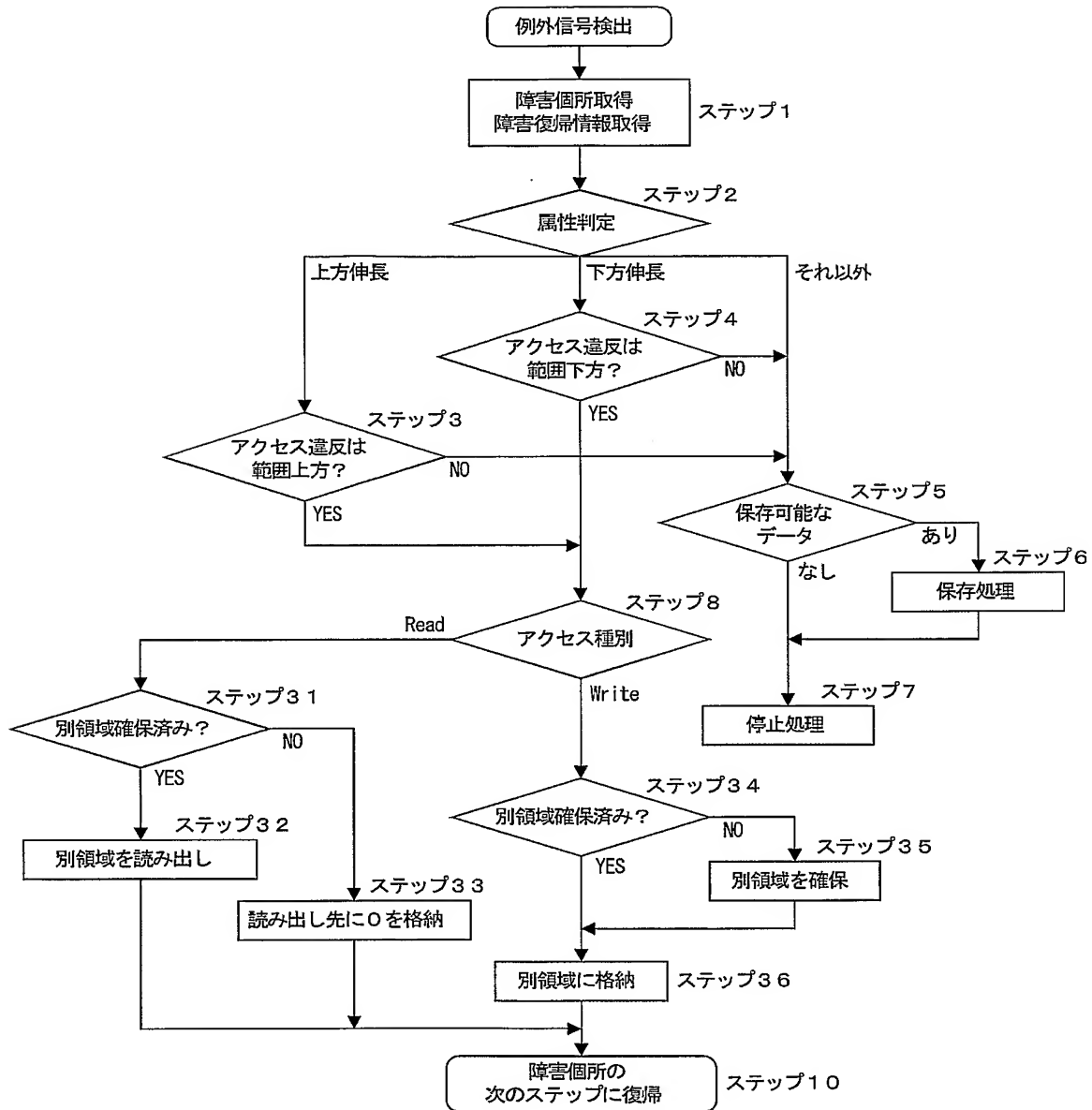
【図 11】



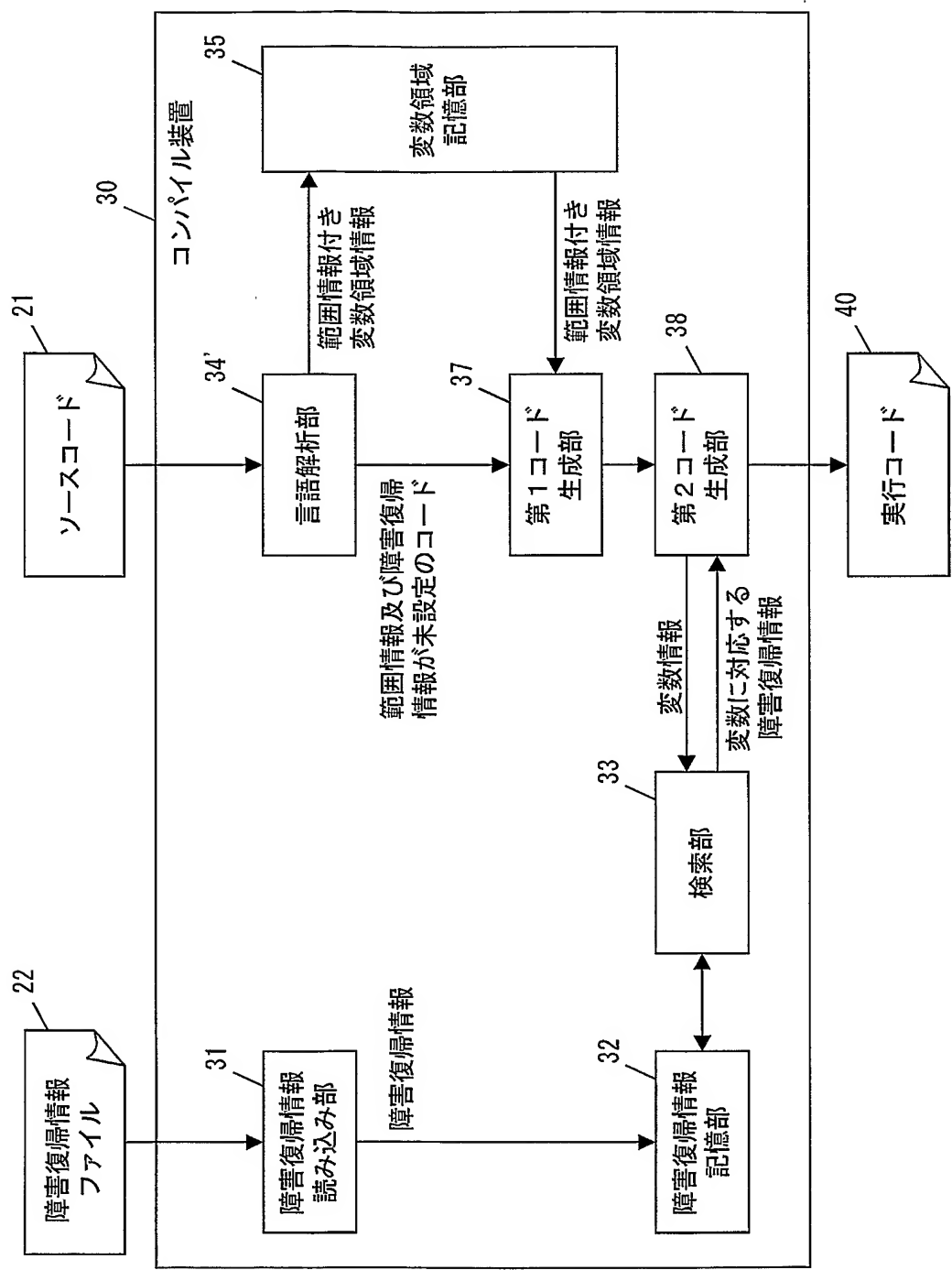
【図 12】



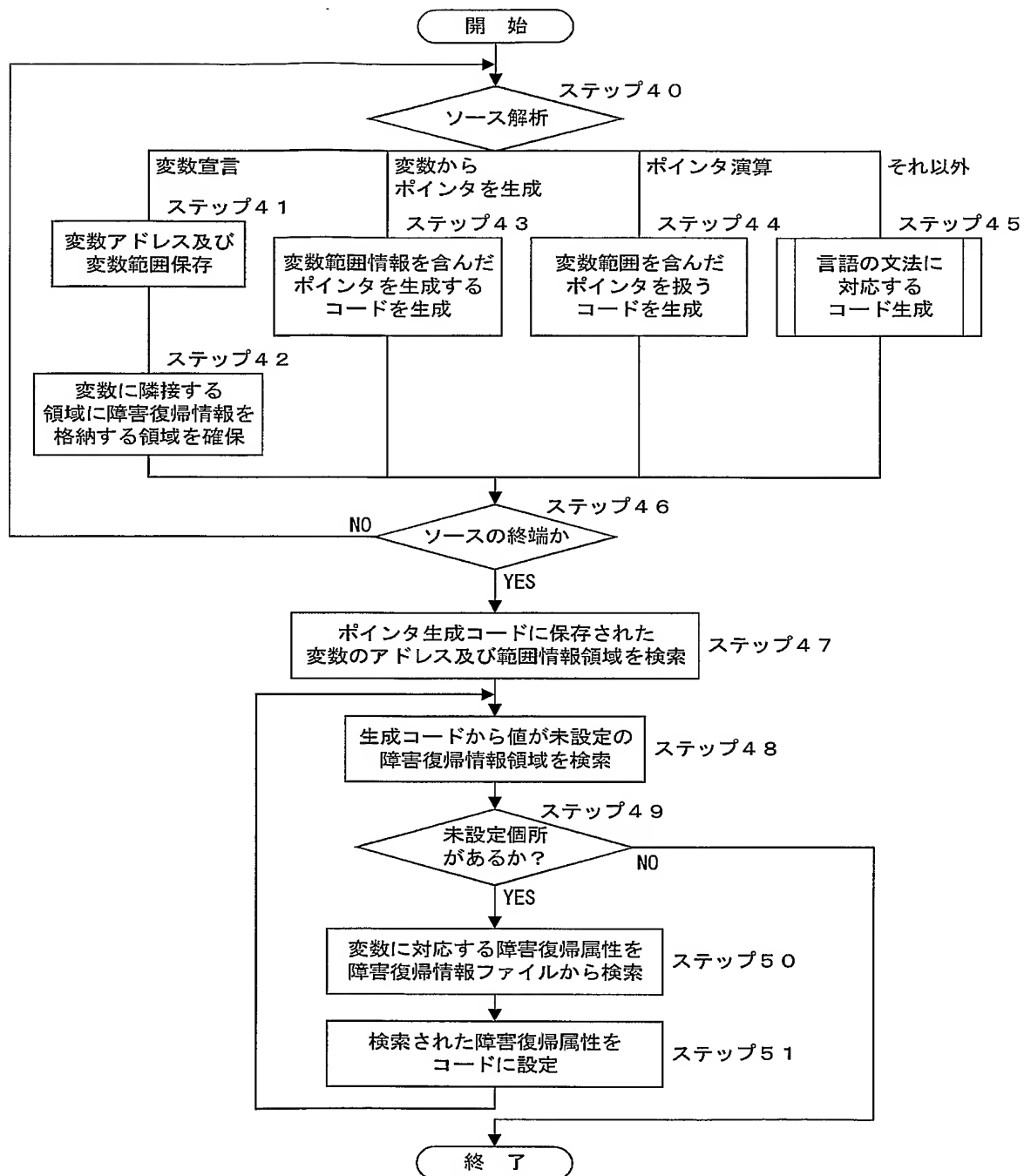
【図 13】



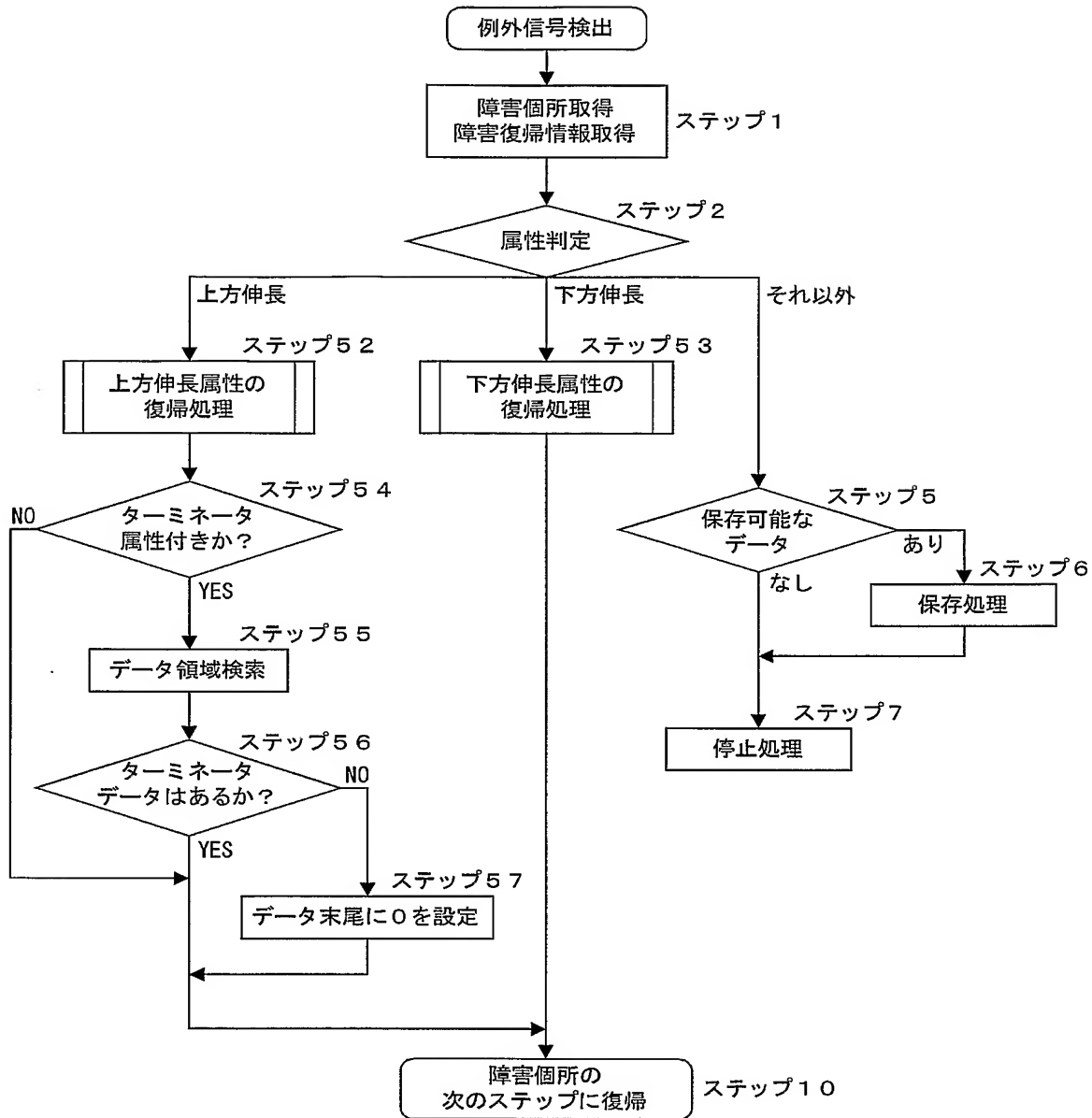
【図 14】



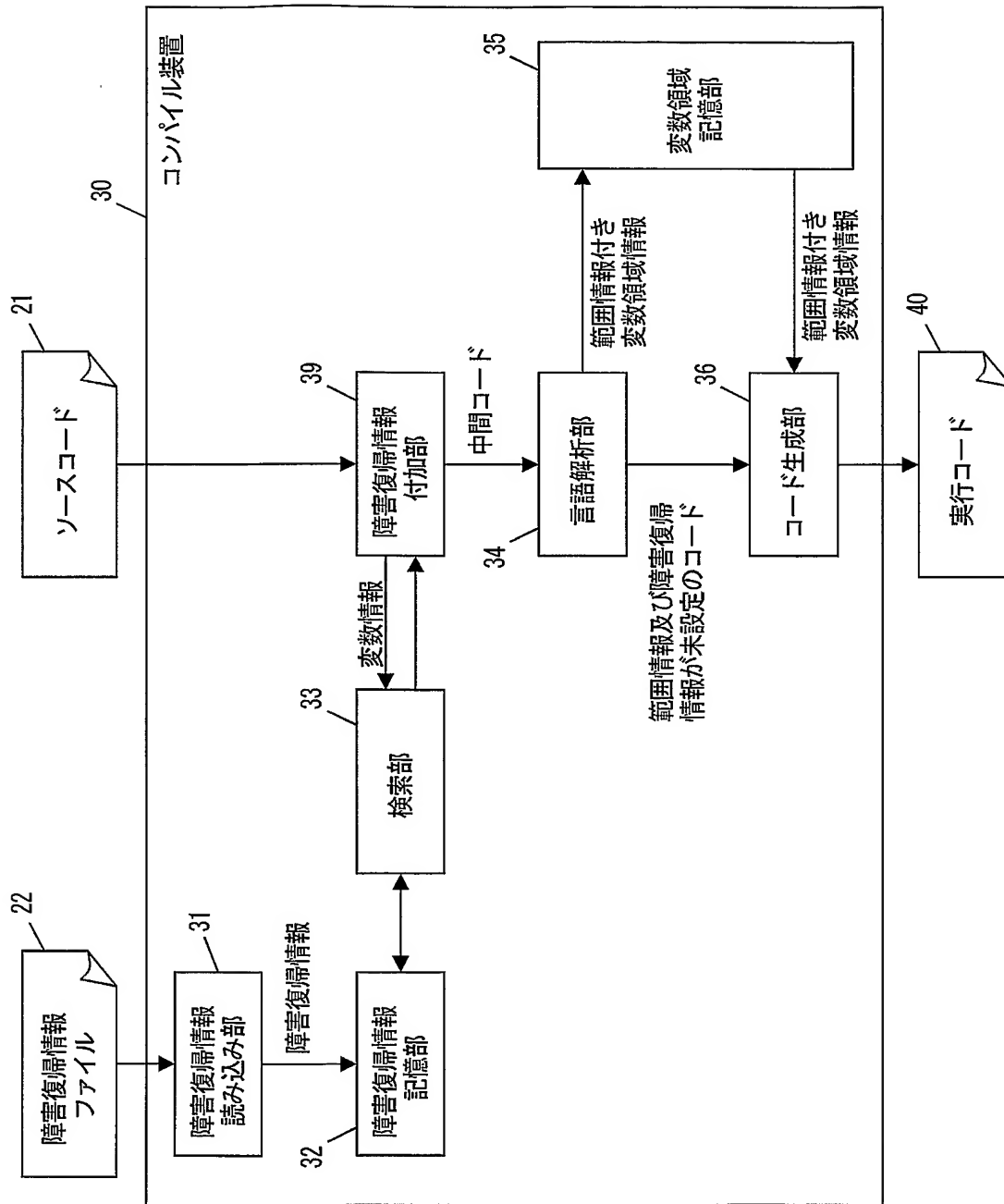
【図 15】



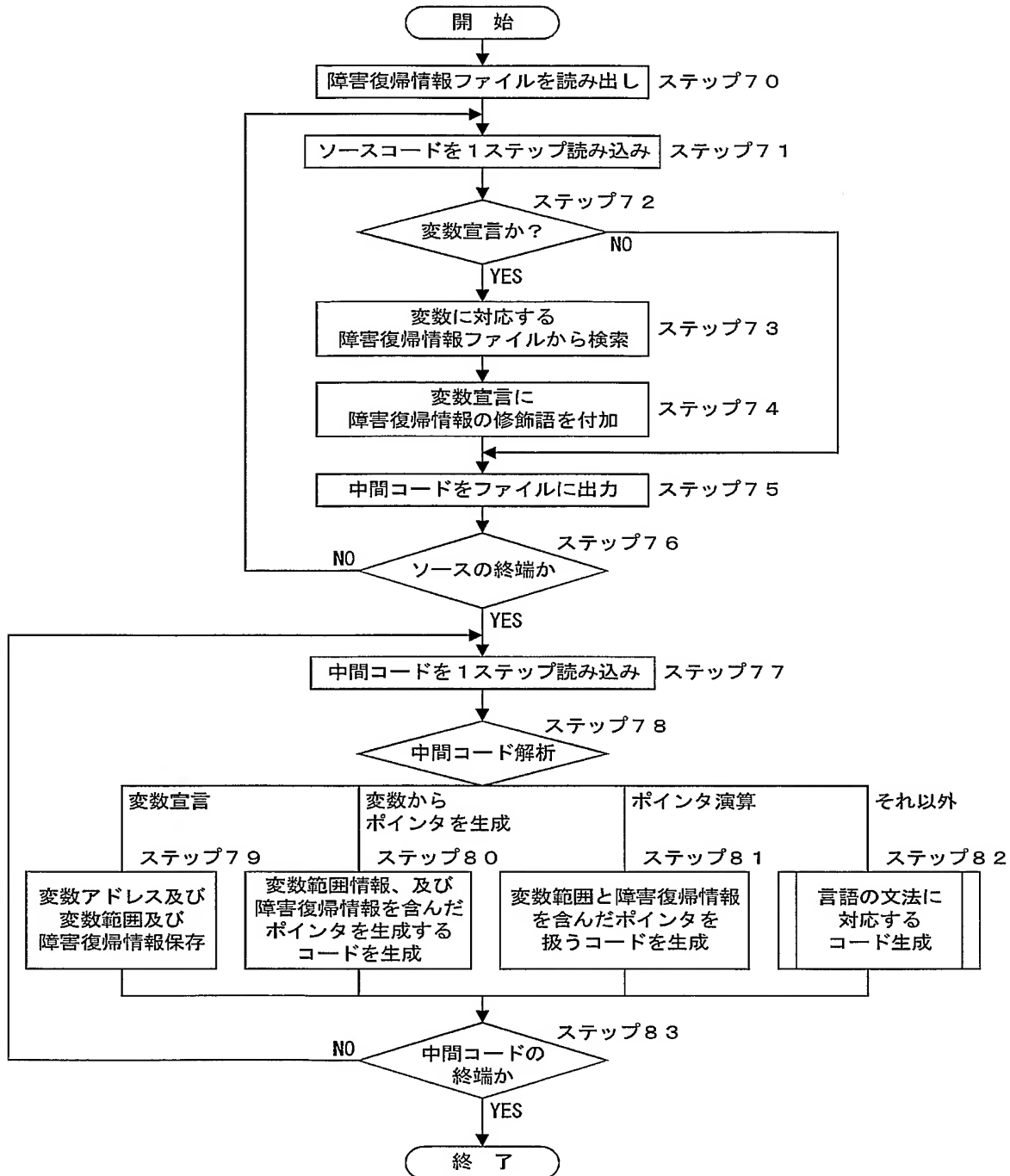
【図 16】



【図 17】



【図 18】





【図 1 9】

(a)

```
01: extern void input_str(char *str);
02:
03: int main()
04: {
05:     UPPER char buf[32];
06:     FIXED int len=0;
07:
08:     input_str(buf);
09:
10:     while ( buf[len] != 0)
11:         len++;
12:
13:     printf( "%d文字です\n", len);
14:
15:     return 0;
16: }
```

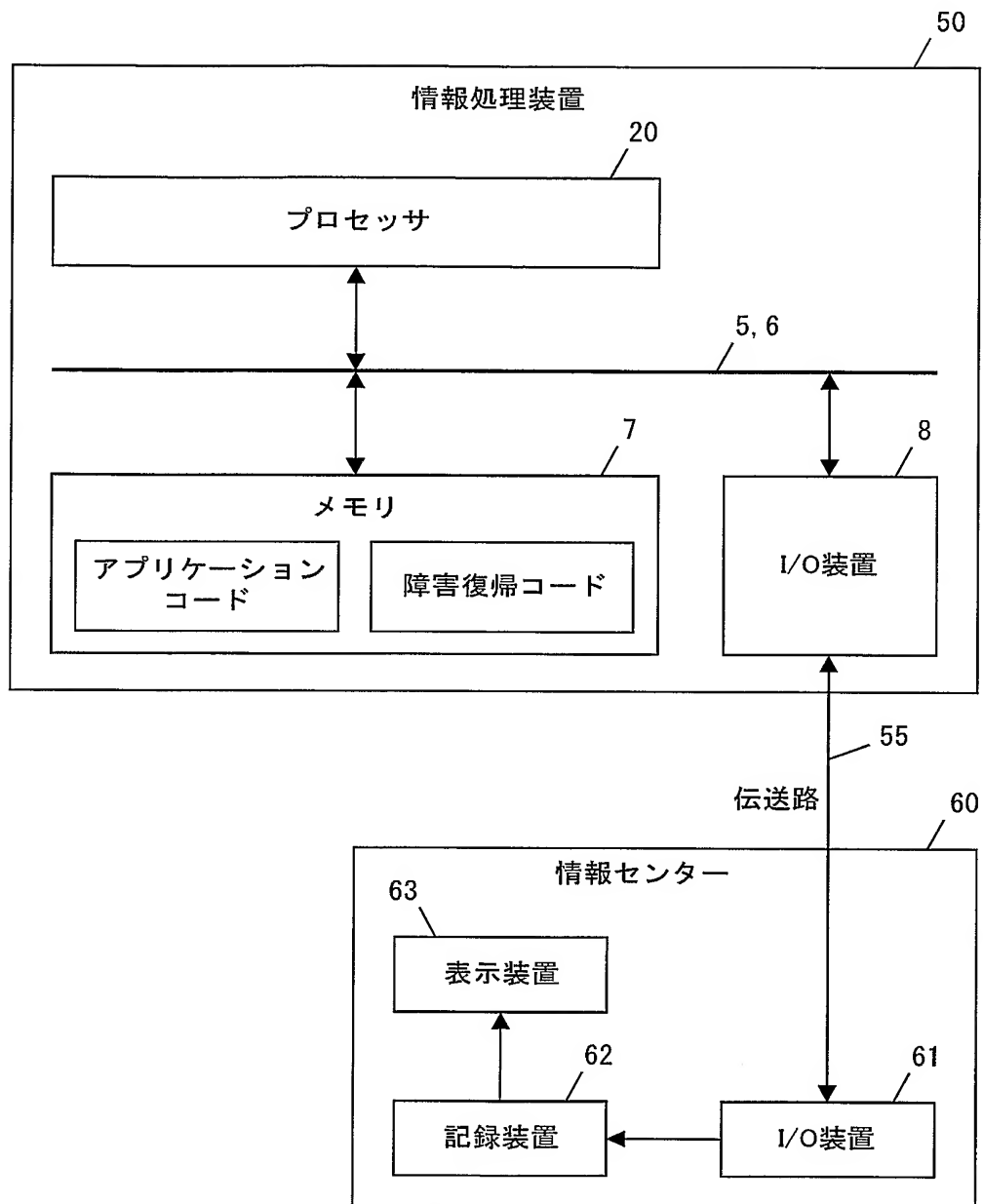
中間コード(main.cの処理後)

(b)

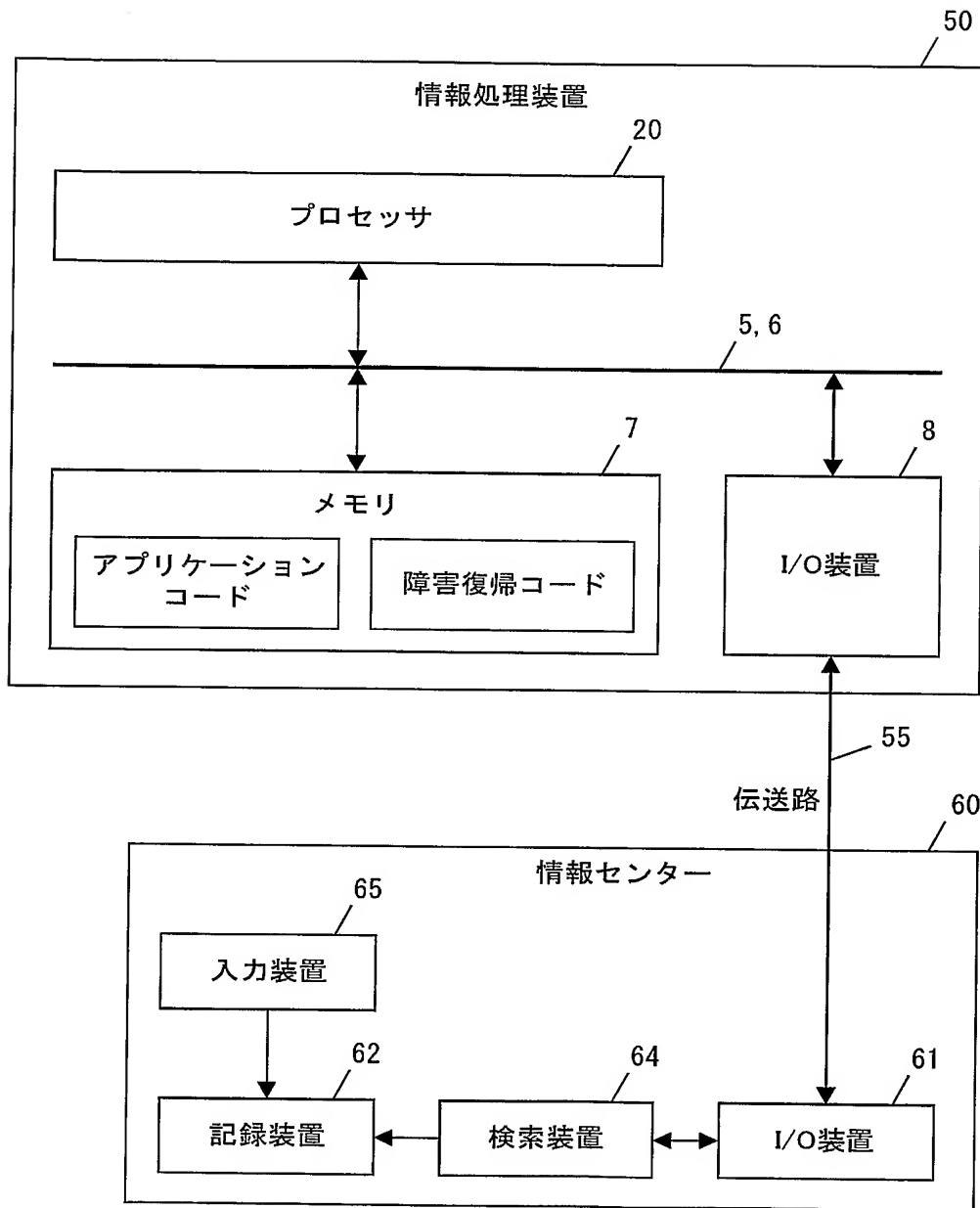
```
01: void input_str(char *str);
02: {
03:     FIXED int c;
04:
05:     while( (c = getchar()) != EOF)
06:         *str++=c;
07: }
```

中間コード(input\_int.c)

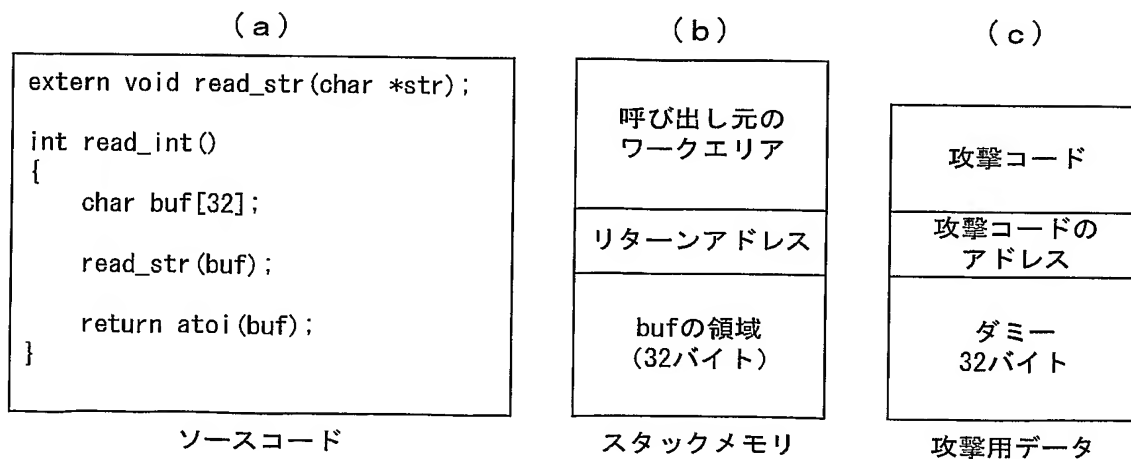
【図 20】



【図 2 1】



【図 2 2】



【書類名】 要約書

【要約】

【課題】 情報処理装置の障害復帰機能を強化する。

【解決手段】 ソースコードのコンパイルの段階で、ポインタ変数に、ポインタの指す変数の範囲情報と、障害復帰用の情報をもたせておく。範囲情報を元にポインタアクセスの不正を検出した際に、障害復帰情報も取得できるので、障害復帰情報に従って、傷害の原因になったデータに適した復帰処理を実施する。範囲情報と障害復帰情報とを、アドレスに関連付けることにより、障害復帰情報（UPPER、LOWER、FIXED）を用いて、障害個所のメモリ領域のデータタイプを特定できる。

【選択図】 図 9

特願 2 0 0 4 - 0 0 8 3 2 7

出 願 人 履 歴 情 報

識別番号

[ 0 0 0 0 0 5 8 2 1 ]

1. 変更年月日

1 9 9 0 年 8 月 2 8 日

[変更理由]

新規登録

住 所

大阪府門真市大字門真 1 0 0 6 番地

氏 名

松下電器産業株式会社